UNITED STATES PATENT AND TRADEMARK OFFICE

Inventors: COLLINS et al. Docket No: 20206-0014(PT-TA-410)

Patent No: **5,848,159**

Issued: December 8, 1998

For: **"PUBLIC KEY CRYPTOGRAPHIC APPARATUS AND METHOD"**

Assistant Commissioner for Patents
Box: Reissue
Washington, D.C. 20231

TRANSMITTAL FOR INFORMATION DISCLOSURE STATEMENT

Enclosed for filing in the above-identified application is an Information Disclosure Statement with attached Form PTO-1449 and copies of cited references.

The Commissioner is authorized to charge any required fees, or credit any overpayment to Deposit Account No. 02-3964 (Order No. 20206-0014(PT-TA-410).

Respectfully submitted,

LEAH SHERRY
Reg. No. 43,918

Dated:

OPPENHEIMER WOLFF & DONNELLY LLP
CUSTOMER NO. 25696
1400 Page Mill Road
Palo Alto, CA 94304
Telephone: 650-320-4000
Facsimile: 650-320-4100

UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor: **Collins et al.**
U.S. Patent No: **5,848,159**
Issue Date: December 8, 1998

Docket No: 20206.14 (PT-TA-410)

For: **"PUBLIC KEY CRYPTOGRAPHIC APPARATUS AND METHOD"**

Assistant Commissioner for Patents
Box: Reissue
Washington, D.C. 20231

INFORMATION DISCLOSURE STATEMENT

Applicants submits herewith the references listed on the attached form PTO-1449 of which Applicants are aware which are believed to be material to the examination of this application and in respect of which there may be a duty to disclose in accordance with 37 CFR 1.56.

The filing of this information disclosure statement shall not be construed as a representation that a search has been made (37 CFR 1.97(g)), nor as an admission that the information cited is, or is considered to be, material to patent ability, nor an admission that no other material information exists.

Respecting for example reference AC, the paper entitled "Using Four-Prime RSA in Which Some of the Bits are specified," Applicants believe that this reference teaches away from the claimed invention. For instance, reference AC does not cover instances where the number of primes is K=3 and K>4. Reference AC merely teaches the extension of 2 prime factors to 4 prime factors for a greater modulus *n*. What is more, the 4 prime factors of *n* are not random but, rather, related through a relationship of the form $p_i=2^k f_i+a_k$. Namely, reference AC teaches a method for determining 4 related primes such that the number of bits required to represent the primes is less than the sum of their length. (See: S.A. Vanstone et al. p. 2118).

The filing of this information disclosure statement shall not be construed as an admission against interest in any manner. Notice of January 9, 1992, 1135 O.G. 13-25, at 25.

DATE: September 27, 2000

Respectfully submitted,

Leah Sherry
Reg. No: 43,918

OPPENHEIMER WOLFF & DONNELLY LLP
1400 Page Mill Road
Palo Alto, CA 94304
Tel: (650) 320-4000
Fax: (650) 320-4100

FORM PTO-1449 U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE INFORMATION DISCLOSURE STATEMENT BY APPLICANT	ATTY DOCKET NO. 20206-0014(PT-TA-410)	PATENT NO. **5,848,159**
	APPLICANT COLLINS et al.	
	ISSUE DATE December 8, 1998	GROUP 2766

U. S. PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING DATE IF APPROPRIATE
	AA	5,761,310	06/1998	Nacıri	380	30	07/18/1996

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	DATE	COUNTRY	NAME	CLASS	SUBCLASS	TRANSLATION YES NO
	AB							

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

	AC	S.A. VANSTONE et al., "Using Four-Prime RSA in Which Some of the Bits are Specified," December 8, 1994, Electronics Letter, Vol. 30, No. 25. pp. 2118-2119
	AD	C. Couvruer et al., "An Introduction to Fast Generation of Large Prime Numbers," 1982, Philips Journal of Research, Vol. 37, Nos. 5-6, pp. 231-264.
	AE	Y. DESMEDT et al., "Public-Key Systems Based on the Difficulty of Tampering (Is There a Difference Between DES and RSA?)," 1986, Lecture Notes in Computer Science, Advances in Cryptology-CRYPTO '86 Proceedings.
	AF	J. J. QUISQUATER et al., "Fast Decipherment Algorithm for RSA Public-Key Cryptosystem" October 1982, Electronic Letters, Vol. 19, No. 21.
	AG	CETIN KAYA KOC, "High-Speed RSA Implementation (Version 2.0)," November 1994, RSA White Paper, RSA Laboratories.
	AH	RIVEST et al., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," February 1978, Communications of the ACM, Vol. 21.
	AI	PKCS #1: RSA Encryption Standard (Version 1.5), November 1993, RSA Laboratories Technical Note.
	AJ	M.O. RABIN, "Digitalized Signatures and Public-Key Functions as Intractable as Factorization," January, 1979, MIT Laboratory for Computer Science.
	AK	R. LIDL et al., "Permutation Polynomials in RSA-Cryptosystems," 1984, Advances in Cryptology—Crypto '83, pp. 293-301.
	AL	D. BONEH et al., "Generating a Product of Three Primes with an Unknown Factorization," Computer Science Department, Stanford University.
	AM	J. J. QUISQUATER et al., "Fast Generation of Large Prime Numbers" June 1982, Library of Congress, Catalog No. 72-179437, IEEE Catalog No. 82CH1767-3 IT, pp. 114-115
	AN	A. J. Menezes et al., "Handbook of Applied Cryptography", 1997, Library of Congress catalog No. 96-27609, pp. 89, 612-613

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if citation considered, whether or not citation is in conformance with MPEP 609; draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

US005761310A

United States Patent [19]

Naciri

[11] **Patent Number:** 5,761,310

[45] **Date of Patent:** Jun. 2, 1998

[54] **COMMUNICATION SYSTEM FOR MESSAGES ENCIPHERED ACCORDING TO AN RSA-TYPE PROCEDURE**

[75] Inventor: Robert Naciri, Chatenay Malabry, France

[73] Assignee: De La Rue Cartes ET Systemes SAS, Paris, France

[21] Appl. No.: 683,493

[22] Filed: Jul. 18, 1996

[30] **Foreign Application Priority Data**

Jul. 26, 1995 [EP] European Pat. Off. 95 09085

[51] Int. Cl.[6] H04L 9/30; H04L 9/00

[52] U.S. Cl. 380/30; 380/9; 380/23; 380/25; 380/49

[58] **Field of Search** 380/9, 23, 24, 380/25, 30, 49, 50

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,405,829	9/1983	Rivest et al.	380/30
4,424,414	1/1984	Hellman et al.	380/30
4,736,423	4/1988	Matyas	380/23
4,870,681	9/1989	Sedlak	380/30
4,933,970	6/1990	Shamir	380/30
4,944,007	7/1990	Austin	380/30 X
5,588,061	12/1996	Ganesan et al.	380/30
5,627,893	5/1997	Demytko	380/30

OTHER PUBLICATIONS

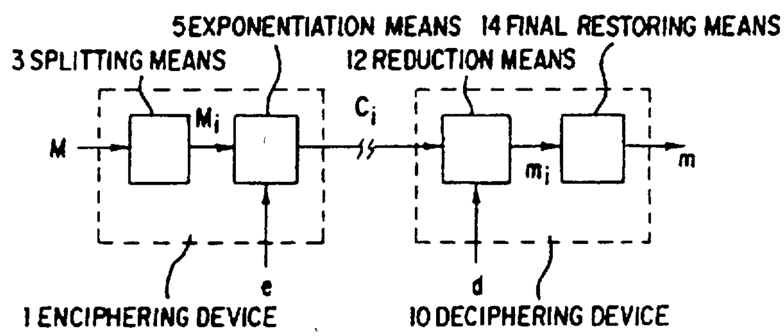
"Fast Decipherment Algorithm for RSA Public-Key Cryptosystem", Electronics Letters 14th Oct. 1982 vol. 18, No. 21, pp. 905–907.

Primary Examiner—Bernarr E. Gregory
Attorney, Agent, or Firm—Oliff & Berridge, PLC

[57] **ABSTRACT**

The procedure involves key numbers "d" and "e" and a modulus N, so hat "N" is the product of two factors "p" and "q" which are prime numbers N=p.q, and $e.d=1_{MOD\phi(N)}$, where $\phi(N)$ is the Euler indicator function. The procedure provides enciphered message parts and for deciphering them comprises: a modulus-determining step for determining a deciphering modulus chosen from "p" and "q", a modular reduction step for making a first modular reduction of the number "d" with a modulus equal to said deciphering modulus "(p−1).(q−1)" with the aim of producing a reduced number, a reduction step for making a second modular reduction of each enciphered message part with a modulus equal to said deciphering modulus with the aim of producing a reduced enciphered message part, an exponentiation step for computing a modular exponentiation of each reduced enciphered message part with a modulus equal to said deciphering modulus and, with an exponent equal to said reduced number with the aim of restoring said message.

5 Claims, 3 Drawing Sheets



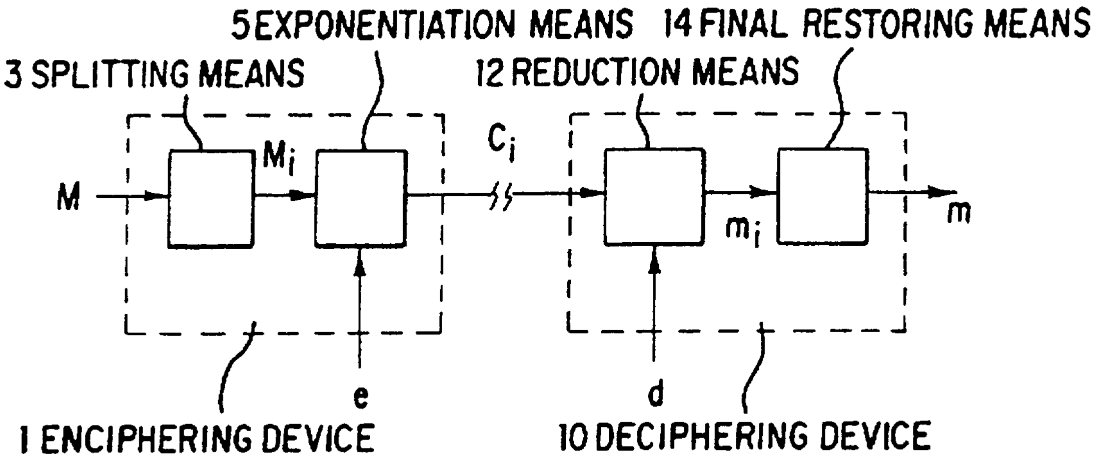

5 EXPONENTIATION MEANS
14 FINAL RESTORING MEANS
3 SPLITTING MEANS
12 REDUCTION MEANS
C_i
M_i
M
m
m_i
e
d
1 ENCIPHERING DEVICE
10 DECIPHERING DEVICE

FIG. 1

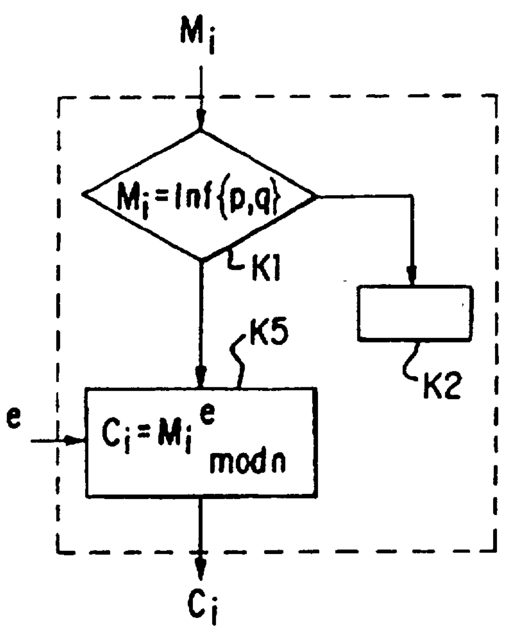

M_i
$M_i = \mathrm{Inf}\{p,q\}$
K1
K5
K2
e
$C_i = M_i^e \bmod n$
C_i

FIG. 2

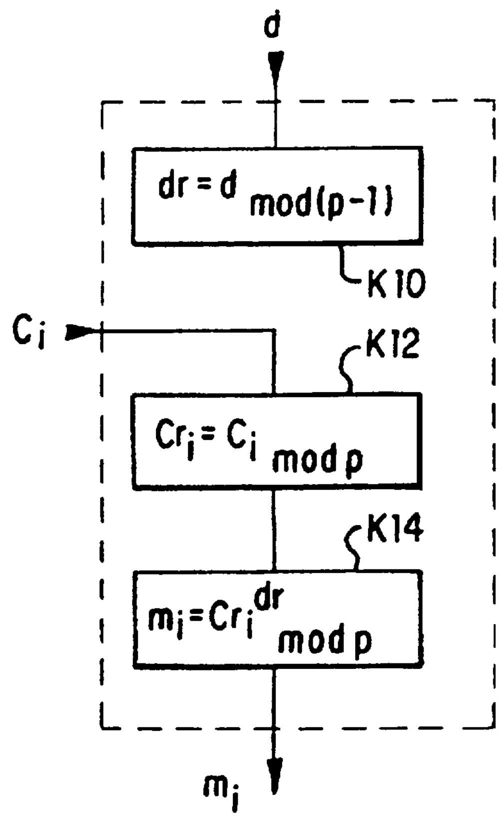


FIG. 3

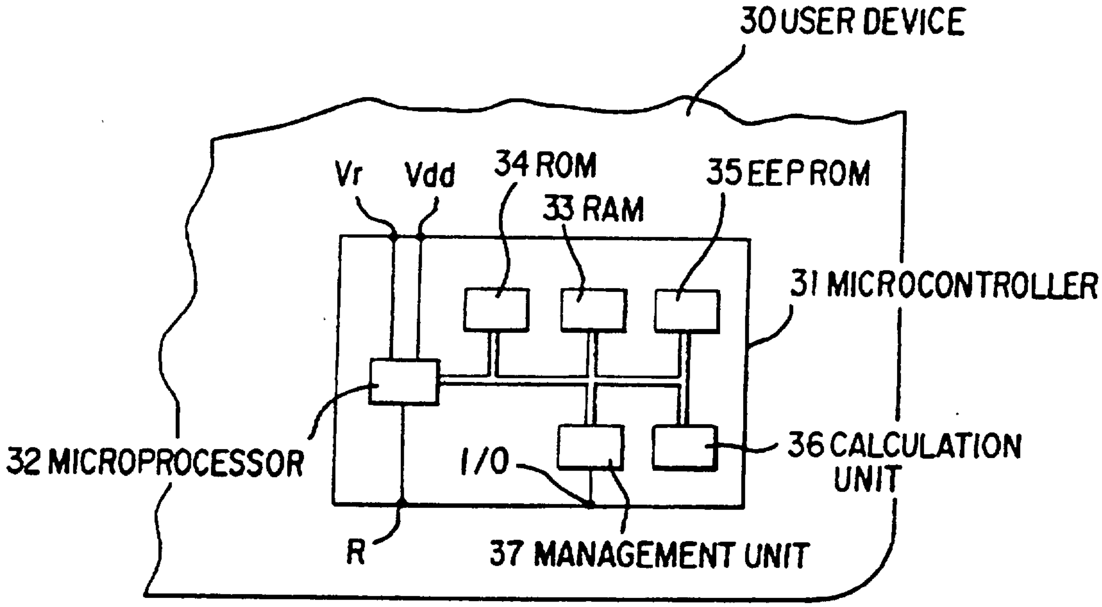

30 USER DEVICE
Vr
Vdd
34 ROM
33 RAM
35 EEPROM
31 MICROCONTROLLER
32 MICROPROCESSOR
I/O
36 CALCULATION UNIT
R
37 MANAGEMENT UNIT

FIG. 4

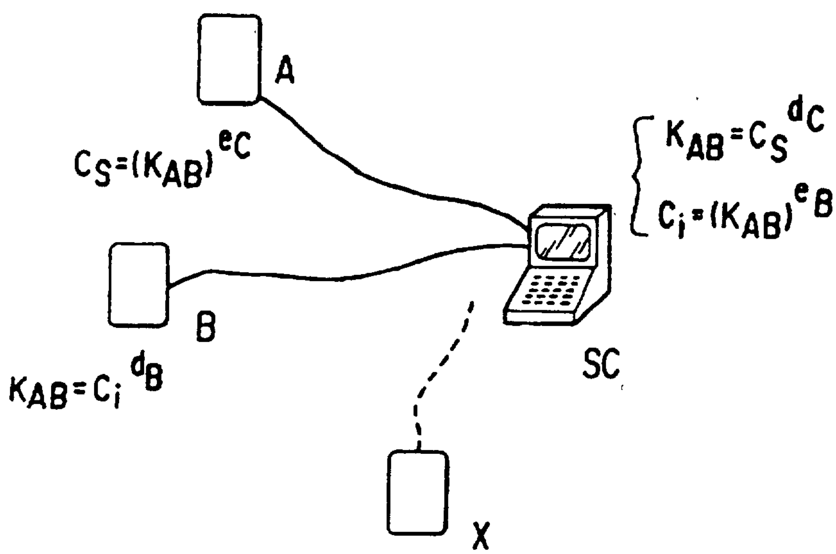

A
$C_S=(K_{AB})^{e_C}$
$K_{AB}=C_S^{d_C}$
$C_i=(K_{AB})^{e_B}$
B
$K_{AB}=C_i^{d_B}$
SC
X

FIG. 5

COMMUNICATION SYSTEM FOR MESSAGES ENCIPHERED ACCORDING TO AN RSA-TYPE PROCEDURE

BACKGROUND OF THE INVENTION

The present invention relates to a communication system for messages enciphered according to an RSA-type procedure which implies key numbers "d" and "e" and a modulus N, so that "N" is a product of two factors "p" and "q" which are prime numbers N=p.q, and $e.d=1_{MOD\phi(N)}$ where $\phi(N)$ is the Euler indicator function, which system comprises, on the one hand, at least an enciphering device formed by:

- splitting means for splitting up the message to be enciphered into at least one message part to be enciphered,
- exponentiation means for carrying out with each message part to be enciphered a modular exponentiation of modulus "N" and having an exponent equal to a first one of said key numbers with the aim of producing a part of the enciphered message, and also at least a deciphering device.

The invention likewise relates to a procedure utilized in the system, a user device of the microcircuit card type comprising on the same medium an enciphering device and a deciphering device and a server center called entrusted center for processing information signals between the various user devices.

A procedure of this type is described in the article entitled "FAST DECIPHERMENT ALGORITHM FOR A PUBLIC-KEY CRYPTOSYSTEM" by J. J. Quisquater and C. Couvreur, published in *ELECTRONICS LETTERS* 14th Oct. 1982.

This procedure implies the use of the Chinese remainder theorem to obtain a rapid deciphering without harming the qualities of the RSA procedure.

SUMMARY OF THE INVENTION

The present invention, also based on the Chinese remainder theorem, proposes a system in which the rapidity of the deciphering process is improved to a very large measure.

Therefore, such a system is characterized in that it comprises at least a deciphering device formed by:

- modulus-determining means for determining a deciphering modulus chosen from said factors,
- first modular reduction means for making a first modular reduction of the number "d" with a modulus equal to said deciphering modulus reduced by unity for producing a reduced number,
- second reduction means for making a second modular reduction of each enciphered message part with a modulus equal to said deciphering modulus with the aim of producing a reduced enciphered message part,
- second exponentiation means for computing a modular exponentiation of each reduced enciphered message part with a modulus equal to said deciphering modulus and with an exponent equal to said reduced number with the aim of restoring said message.

Thus, due to the measures recommended by the invention, it is no longer necessary to perform the combining operation of the remainders of formula (1) of aforementioned article.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other aspects of the invention will be apparent from and elucidated with reference to the embodiments described hereinafter.

In the drawings:

FIG. 1 shows a communication system according to the invention,

FIG. 2 shows an enciphering flow chart in accordance with the invention,

FIG. 3 shows a deciphering flow chart according to the invention,

FIG. 4 shows the scheme of a user device, and

FIG. 5 shows a system according to the invention, which implies a server called confidence server and a plurality of user devices.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In FIG. 1, reference 1 indicates the enciphering device. This one receives a message M, for example the French word "BONJOUR" which means GOOD-DAY. This message is split up by the splitting means 3 into message parts to be enciphered. These parts are formed each by letters forming the part and a sequence of digital codes is obtained, for example, the decimal digital codes M=66, M2=79, M3=78, M4=74, M5=79, M6=85, M7=82, which represent the ASCII codes for "BONJOUR". Exponentiation means 5 perform exponentiations of these digital codes by taking parameters "e" and "N" in accordance with measures of which the first one results directly from the invention:

- Numbers p and q are taken to be higher than 255: p=263 and q=311, so that N=p.q=81793.
- "e" is selected, so that it is a prime number with p−1 and q−1, that is: e=17.

Now one determines d: $e.d=1_{MOD\phi(N)}$.

In the case where p and q are prime numbers $\phi(N)=(p-1).(q-1)$ that is: $e.d=1_{MOD(40610)}$. Among the "d", which satisfy the above relation, one may take 54943. In principle, "d" is unknown at the enciphering device 1. The means 5 may encode the message by computing the modular exponentiation of each of said codes:

$$Ci=Mi^{17}{}_{MOD(N)}\ i=1\ldots7$$

giving the coded message which comprises the enciphered parts:

C1=62302, C2=47322, C3=74978, C4=00285, C5=47322, C6=09270, C7=54110.

According to the invention, for deciphering this message, a deciphering device 10 is provided. This device uses a first means for determining the deciphering modulus from the numbers "p" and "q"; preferably, the smaller of the two is chosen to gain on the calculations that is:

$$p=(263),$$

second means perform the modular reducing operation with a number "d"

$$\begin{aligned} d_r &= 5493_{MOD262} \\ &= 185. \end{aligned}$$

Then, depending on the modulus "p", the enciphered message parts are reduced.

$$\begin{aligned} C_r1 &= 62302_{MOD263} \\ &= 234 \end{aligned}$$

and respectively, $C_r2=245$, $C_r3=023$, $c_r4=022$, $C_r5=245$, $c_r6=065$, c_r7 giving the deciphered message parts:

$m_1 = C_r 1^{185}{}_{MOD263} = 66$

$m_2 = c_r 2^{185}{}_{MOD265} = 79$

$m_3 = C_r 3^{185}{}_{MOD263} = 78$

$m_4 = C_r 4^{185}{}_{MOD263} = 74$

$m_5 = c_r 5^{185}{}_{MOD265} = 79$

$m_6 = c_r 6^{185}{}_{MOD265} = 85$

$m_7 = C_r 7^{185}{}_{MOD263} = 82.$

The message "m" is then restored by concatenation, by the final restoring means 4 transcribing in usual characters. If everything is done well. m=M.

The enciphering device 1 and the deciphering device 10 are actually put into effect based on processors programmed for executing the operations of the flow charts shown in the FIGS. 2 and 3 which follow.

The flow chart of FIG. 2 explains the operation of the device 1. Box K1 indicates a test made with each part of the enciphered message. If this value exceeds that of the selected deciphering modulus (thus the smaller of "p" and "q"), then there is declared that there is an error in box K2. If not, box K5 is proceeded to, where the actual enciphering operation is carried out, that is to say, a modular exponentiation. Thus, enciphered message parts Ci are obtained.

The flow chart of FIG. 3 shows the deciphering operations carried out by the deciphering device 10. This flow chart shows in box K10 an operation prior to the reduction of the number d to obtain a reduced key number "dr". Box K12 is a modular reducing operation of "p", carried out with the enciphered message parts, and box K14 is a modular exponentiation of modulus "p" and whose exponent is "dr" with the enciphered message parts.

The enciphering and deciphering devices may be inserted on the same medium to form a user device. The devices can then communicate with each other by utilizing the enciphering procedure according to the invention.

FIG. 4 shows the structure of a user device. This device is made on the basis of a trusted microcontroller such as, for example, the 83C852 made by Philips. Such a microcontroller 31 is shown in FIG. 4 and formed by a microprocessor 32, a random access memory 33 and a read-only memory 34 which notably contains instructions of operation for implementing the invention, notably the enciphering operations and deciphering operations already described. It also comprises an EEPROM memory 35 for containing various data such as the secret key of the card, the public key of a third party with which it exchanges information signals . . . It also comprises a calculation unit 36 which carries out the necessary operations for the functions of enciphering, a management unit 37 for the inputs/outputs furthermore connected to an input I/O of the microcontroller 31. Said elements of the microcontroller 31 are interconnected by a bus 38.

Any additional detail may be found back in the manual of the microcontroller 83C852 mentioned above.

An interesting example of an application of the invention is the transfer of key DES by the RSA, as this has been described in the article "Threats of Privacy and Public Keys for Protection" by Jim Bidzos, published in the document PROCEEDINGS OF COMPCON, 91, 36TH IEEE COMPUTER SOCIETY INTERNATIONAL CONFERENCE, 25 Feb. to 1 Mar. 1991, San Francisco-N.Y. (U.S.). The protocol for exchanging session key DES between two users A and B via a public channel may be as follows. Let $\{n_A, d_A\}$ and $\{n_B, d_B\}$ be the respective secret keys of the users A and B. For example, A wishes to transmit the session key K_S to B.

User A

$m = K_s$

The enciphering is carried out by using the public key e_B.

$C_{AB} = m^{e}B$

and the cryptogram C_{AB} is transmitted by the public channel.

User B

reception of the cryptogram $m = C_{AB}$

deciphering of the cryptogram $m = C_{AB}{}^{d}B$

the use of key K_S

Generally, K_S is approximately smaller than a key RSA. K_S may be a session key DES of the order of 56 binary elements and p and q are of the order of 256 bits.

Lots of official bodies prohibit the enciphering of messages in public communications. The invention applies particularly well when a confidence server SC is used to avoid this prohibition.

This case is shown in FIG. 5. This Figure illustrates the case where a plurality of user devices A, B, . . . X can communicate with each other via a confidence server SC. This server SC has all the knowledge to know all the messages exchanged by the various users uncoded.

By way of example there is explained the case where the user A wishes to communicate a key DES K_{AB} to user B. Thus, A enciphers the key K_{AB} with the aid of the public key e_s of the server SC which itself deciphers same upon reception, then enciphers it with B's public key e_B. Finally, B deciphers the cryptogram with the aim of finding back the key which was initially sent by A.

Thus, in this case the procedure is applied both at the deciphering end of the user B and that of server SC. Thus, the invention is used to obtain a good availability of the server for a plurality of users. The gain of computation caused by the invention is particularly noticeable.

What is claimed is:

1. Communication system for messages enciphered according to an RSA-type procedure which implies key numbers "d" and "e" and a modular number N, so that "N" is a product of two factors "p" and "q" which are prime numbers N=p.q and that $e.d = 1_{MOD\phi(N)}$ where $\phi(N)$ is the Euler indicator function, which system comprises, on the one hand, at least an enciphering device formed by:

splitting means for splitting up the message to be enciphered into at least one message part to be enciphered,

exponentiation means for carrying out with each message part to be enciphered a modular exponentiation of modulus "N" and having an exponent equal to a first one of said key numbers with the aim of producing a part of the enciphered message, and on the other hand at least a deciphering device, characterized in that it comprises at least a deciphering device formed by:

modulus determining means for determining a deciphering modulus chosen from said factors,

first modular reduction means for making a first modular reduction of the number "d" with a modulus equal to said deciphering modulus reduced by unity for producing a reduced number,

second reduction means for making a second modular reduction of each enciphered message part with a modulus equal to said deciphering modulus with the aim of producing a reduced enciphered message part,

second exponentiation means for computing a modular exponentiation of each reduced enciphered message part with a modulus equal to said deciphering modulus

and with an exponent equal to said reduced number with the aim of restoring said message.

2. Enciphering/deciphering procedure utilized in the system as claimed in claim 1, according to which procedure, and for enciphering a message:

said message is split up into message parts to be enciphered,

each part undergoes a modulo-M modular exponentiation operation with an exponent equal to a first one of said key numbers, for producing enciphered message parts, and for deciphering the message:

the enciphered message parts undergo a deciphering exponentiation operation for producing deciphered message parts, characterized in that

the message parts to be enciphered are presented in the form of numbers smaller than the numbers p and q,

the deciphering exponentiation operation comprises:

a step for determining a deciphering modulus chosen from said factors,

a preceding step for making a first modular reduction of the number "d" with a modulus equal to said deciphering modulus reduced by unity with the aim of producing a reduced number,

a step for making a second modular reduction of the parts of the enciphered messages with a modulus equal to said deciphering modulus for producing reduced enciphered message parts,

a modular exponentiation step made with the parts of the reduced enciphered messages with a modulus equal to said deciphering modulus and with an exponent equal to said reduced number.

3. User device for a communication system in which messages are enciphered according to an RSA-type procedure which implies key numbers "d" and "e" and a modular number N, so that "N" is a product of two factors "p" and "q" which are prime numbers N=p.q and that $e.d=1_{MOD\phi(N)}$ where $\phi(N)$ is the Euler indicator function said user device comprising an enciphering device formed by:

splitting means for splitting up the message to be enciphered into at least one message part to be enciphered,

exponentiation means for computing with each message part to be enciphered a modular exponentiation of modulus "N" and having an exponent equal to a first one of said key numbers, with the aim of producing a part of the enciphered message, and at least a deciphering device, characterized in that the enciphering device is formed by:

modulus determining means for determining a deciphering modulus chosen from said factors,

first modular reduction means for making a first modular reduction of the number "d" with a modulus equal to said deciphering modulus reduced by unity for producing a reduced number,

second reduction means for making a second modular reduction of each enciphered message part with a modulus equal to said deciphering modulus with the aim of producing a reduced enciphered message part,

second exponentiation means for effecting a modular exponentiation of each reduced enciphered message part with a modulus equal to said deciphering modulus and with an exponent equal to said reduced number to restore said message.

4. User device as claimed in claim 3, wherein said user device comprises a chip card.

5. Server for a communication system for messages enciphered according to an RSA-type procedure which implies key numbers "d" and "e" and a modular number N, so that "N" is a product of two factors "p" and "q" which are prime numbers N=p.q and that $e.d=1_{MOD\phi(N)}$ where $\phi(N)$ is the Euler indicator function, said server comprising an enciphering device and a deciphering device for using intermediaries with user devices, said enciphering device formed by:

splitting means for splitting up the message to be enciphered into at least one message part to be enciphered,

exponentiation means for computing with each message part to be enciphered a modular exponentiation of modulus "N" and having an exponent equal to a first one of said key numbers, with the aim of producing a part of the enciphered message,

and characterized in that said deciphering device is formed by:

modulus determining means for determining a deciphering modulus chosen from said factors,

first modular reduction means for making a first modular reduction of the number "d" with a modulus equal to said deciphering modulus reduced by unity with the aim of producing a reduced number,

second reduction means for making a second modular reduction of each enciphered message part with a modulus equal to said deciphering modulus with the aim of producing a reduced enciphered message part,

second exponentiation means for computing a modular exponentiation of each reduced enciphered message part with a modulus equal to said deciphering modulus and with an exponent equal to said reduced number to restore said message.

* * * * *

Electronics Letters Online No. 19941450

P.-L. Lin and J. G. Dunham (*Southern Methodist University, Dallas, TX 75275, USA*)

References

1 BEN-OR, M., GOLDWASSER, S., and WIDGERSON, A.: 'Completeness theorems for non-cryptographic fault-tolerant distributed computation'. Proc. 20th STOC, 1988, (ACM), pp. 1–10

2 BLAKLEY, G.R.: 'Safeguarding cryptographic keys'. Proc. AFIPS 1979 National Computer Conf., 1979, Vol. 48, pp. 313–317

3 GALLAGER, R.G.: 'Information theory and reliable communications' (Wiley, New York, 1968)

4 RABIN, T., and BEN-OR, M.: 'Verifiable secret sharing and multiparty protocols with honest majority'. Proc. 21st STOC, 1989, (ACM), pp. 73–85

5 SHAMIR, A.: 'How to share a secret', *Commun. ACM*, 1979, 22, (11), pp. 612–613

6 SHANNON, C.: 'Communication theory of secrecy systems', *Bell Syst. Technol. J.*, 1949, 28, pp. 565–715

Using four-prime RSA in which some of the bits are specified

S.A. Vanstone and R.J. Zuccherato

Indexing term: Cryptography

In the Letter the authors apply their method of predetermining a certain number of bits of the RSA public key modulus to RSA using four primes. The method works just as well using four primes as it does with two, and does not appear to decrease the security level. It appears that at most 252 bit can be predetermined, whereas in the two-prime case half of the bits could be specified.

Introduction: Traditionally, the RSA cryptosystem [1] has been used with two primes and a modulus of 512bit. Recent advances in factoring, however, have made 512bit RSA dangerously close to insecure. For this reason, the use of 1024bit RSA is now recommended. It may be of some advantage to use the same database of primes for 1024bit RSA as was used for 512bit RSA. This would mean using a 1024bit RSA with four primes. Also, it would be desirable to be able to predetermine some of these bitbit to effectively reduce the key size for transmission and storage purposes as was done for two-prime RSA in [2, 3]. This would be particularly advantageous where a group of users use the same t bit as the high- and low-order bit of their public key modulus. Then only $1024-t$ bit need to be stored for each user and one copy of the t bit for the entire group. There may be situations where a user would like the t bit to be a binary representation of their user ID and other publicly available information. This situation can also be implemented. We show that one can always specify up to 252 bit of a 1024bit modulus using four primes.

Using four-primes RSA: In this version of RSA, four primes of 256 bit are generated. Let these primes be p_i for $i = 1, 2, 3, 4$. Then a random e together with n is taken as the public key, with $d_i = e^{-1}(\text{mod } (p_i-1))$ being the private keys. The message M is encrypted as $C = M^e \pmod n$. Decryption is accomplished by calculating $M_i = C^{d_i}(\text{mod } p_i)$ for $i = 1, 2, 3, 4$ and combining the results using the Chinese remainder theorem. This causes a doubling of the decryption time over 512bit RSA, compared with an increase by a factor of 4 using conventions 1024bit RSA.

Using four-prime RSA would give the added security advantage of using a 1024bit modulus. The only factoring algorithm that may be more effective on a four-prime modulus than on the usual modulus using two primes is the elliptic curve factoring algorithm [4]. Using known running times, it appears that the number field sieve would require about 10^{29} operations to factor such a number, while the elliptic curve algorithm would use about 10^{14} operations. This appears to be infeasible because to date, the elliptic curve method has been unsuccessful ... finding factors of greater than 40 digits [5, 6], and here the primes are about 80 digits. The use of four-prime RSA also allows one to use the same database of primes as was used in 512bit RSA, and does not increase decryption time as much as using two-prime 1024bit moduli.

Specifying some of the bits. Let f_i be a $c$bit integer and a_i be l bit for $i = 1, 2, 3, 4$. Now, if we let $n = \prod_{i=1}^{4} p_i$ where $p_i = 2^k f_i + a_i$, then we obtain

$$
\begin{aligned}
n &= 2^{4k} f_1 f_2 f_3 f_4 \\
&\quad + 2^{3k}(f_1 f_2 f_3 a_4 + f_1 f_2 a_3 f_4 + f_1 a_2 f_3 f_4 + a_1 f_2 f_3 f_4) \\
&\quad + 2^{2k}(f_1 f_2 a_3 a_4 + f_1 a_2 f_3 a_4 + f_1 a_2 a_3 f_4 + a_1 f_2 f_3 a_4 \\
&\qquad + a_1 f_2 a_3 f_4 + a_1 a_2 f_3 f_4) \\
&\quad + 2^{k}(f_1 a_2 a_3 a_4 + a_1 f_2 a_3 a_4 + a_1 a_2 f_3 a_4 + a_1 a_2 a_3 f_4) \\
&\quad + a_1 a_2 a_3 a_4
\end{aligned}
$$

We would like n to be 1024 bit, so we must have $4k + 4c = 1024$. We would also like the product $f_1 f_2 f_3 f_4$ to be specified ahead of time and visible as the top $4c$ bit of n. To allow this, we would require the remaining terms be less than $4k$ bit. In other words, $3k + 3c + l + 3 < 4k$. To make the product $a_1 a_2 a_3 a_4$ difficult to obtain we need at least 60 bit of rippling into the 2^k term. Thus $k + 60 < 4l$. Now we want to maximise the number of bit that can be specified ahead of time. To do this we want to maxmise the size of the product $f_1 f_2 f_3 f_4$ and the number of bit of $a_1 a_2 a_3 a_4$ that are not hidden. Thus we want to maximise $4c + k$. Solving this integer linear program we obtain $k = 210$, $l = 68$ and $c = 46$. This gives at most 394 bit of the product that can be specified ahead of time. As is shown below, to generate these primes effectively, at most l low-order bit should be specified. With the above parameters this gives at most 252 bit of the product that can always be specified. To make this generation feasible, we would also require that ~20 bit be available to search the residue class, so we should only specify about 48 low-order bit, giving 232 bit that could be predetermined.

This could be accomplished by first choosing random a_1, a_2 and a_3 until p_1, p_2 and p_3 are prime. We are assuming here that f_1, f_2, f_3 and f_4 are public knowledge and predetermined. If we wish the last 48 bit of n to be α, we then solve the congruence $a_1 a_2 a_3 \gamma \equiv \alpha \pmod{2^{48}}$ for γ. Because γ is 48 bit long and we require $a_4 \equiv \gamma \pmod{2^{48}}$, where a_4 is 68 bit long, we can search this residue class until we obtain a p_4 that is prime.

Using this as a base case, all four schemes presented in [2] can be realised. We believe that this is the largest number of bits that can be predetermined, generalising the ideas presented.

Security issues: As mentioned in [2], predetermining some of the bits of an RSA modulus appears to be as secure as using a general modulus. Using four primes, however, gives much smaller prime factors and may allow the elliptic curve factoring method to be more of an attack. As mentioned above, we do not believe this should be a major concern. None of the other known factoring algorithms appears to be able to factor numbers of this size, regardless of the fact that they have four primes or the special structure imposed by prespecifying some of the bits.

Two additional attacks were mentioned in the original paper. They were not feasible for 1024bit moduli. After generalisation to four primes, they do not appear to even apply. For this reason, there may even be an increase in security in using four-prime RSA over two prime RSA and predetermining some of the bits.

Conclusion: It appears that, when predetermining some of the bits of an RSA modulus, it may be worthwhile to use four primes instead of the usual two.

 16 August 1994

Electronics Letters Online No: 19941466

S. A. Vanstone and R. J. Zuccherato (*Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada*)

References

1 RIVEST, R. ... digital si... 1978, 21, ...
2 VANSTONE ... generation ...
3 VANSTONE ... UK Pater...
4 LENSTRA, ... 126, pp. 6...
5 COHEN, H ... (Springer- ...
6 VAN OOR... cryptosyst... logarithm... The scien...

Combin... maser fr...

M.E. Cost...
D.G. Blai...

Inde... Me...

An ... star ... of a ... ma... hyd... inte... one ... out ...

Introductio... developed ... long-term ... phire oscill... have achie... times [2] a... atory have ...

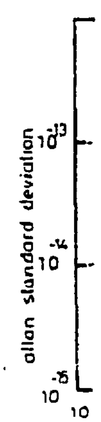


Fig. 1 *Fr... tralia's sa...*

—— m
– – – sa

This L... standard ... short-term ... bility of ... this a V... (tightly) ...

References

1 RIVEST, R.L., SHAMIR, A., and ADLEMAN, L.: 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, 1978, 21, (2), pp. 120–126
2 VANSTONE, S.A., and ZUCCHERATO, R.J.: 'Short RSA keys and their generation', *to be published in J. Cryptol.*
3 VANSTONE, S.A., and ZUCCHERATO, R.J.: 'Key transmission system', UK Patent Application
4 LENSTRA, H.W.: 'Factoring with elliptic curves', *Ann. Math.*, 1987, 126, pp. 649–673
5 COHEN, H.: 'A course in computational algebraic number theory' (Springer-Verlag, Berlin, 1993)
6 VAN OORSCHOT, P.C.: 'A comparison of practical public key cryptosystems based on integer factorization and discrete logarithms', *in* SIMMONS, G.J. (Ed.): 'Contemporary cryptology – The science of information integrity' (IEEE Press, 1991)

Combined sapphire oscillator – hydrogen maser frequency standard

M.E. Costa, J.W. He, A.S. Mann, A.N. Luiten and D.G. Blair

Indexing terms: Oscillators, Masers, Frequency measurement, Measurement standards

An attempt has been made to create a superior frequency standard by combining the excellent short term frequency stability of a sapphire oscillator with the long-term stability of a hydrogen maser. The combined frequency standard closely follows the hydrogen maser in the long term and has good stability at short integration times, although it falls short of the actual sapphire oscillator stability due to phase noise in the hydrogen maser output signal.

Introduction: A variety of high stability oscillators have been developed that have excellent performance in either the short- or long-term domains [1]. In particular, as shown in Fig. 1, the sapphire oscillators developed at the University of Western Australia have achieved a frequency stability of 3×10^{-15} at short integration times [2] and the hydrogen masers developed at Shanghai Observatory have a stability of $3–6 \times 10^{-15}$ at long integration times [3].

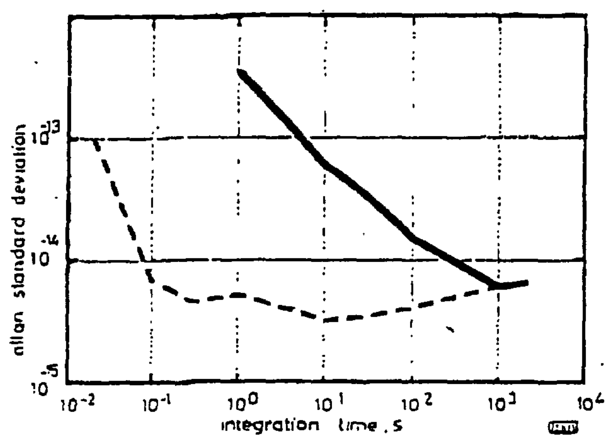


Fig. 1 *Fractional frequency stability of University of Western Australia's sapphire oscillator and Shanghai Observatory's hydrogen maser*

—— maser
- - - - sapphire oscillator

This Letter reports an attempt to create a superior frequency standard by combining in a 5MHz quartz crystal the excellent short-term stability of a sapphire oscillator and the long-term stability of the Shanghai Observatory Hydrogen maser. To achieve this a Vectron CO-246 quartz crystal oscillator is broadband (tightly) phaselocked to the sapphire oscillator and narrowband (loosely) locked to the hydrogen maser. The combined frequency standard has hydrogen maser stability at long integration times and is at the 10^{-14} level at short integration times. This falls short of the sapphire oscillator stability but is a substantial improvement over the normal hydrogen maser short-term stability.

Combining the frequency standards: Fig. 2 is a simplified schematic diagram of the combined sapphire oscillator–hydrogen maser frequency standard. In the broadband phaselocked loop the 5MHz signal from the quartz crystal is multiplied to 320MHz and then applied to a step recovery diode to generate a comb of harmonics extending to microwave frequencies. These are used to heterodyne the 11.93GHz signal from the sapphire oscillator down to about 11MHz, which is then mixed with two HP3325 function generators (locked to the crystal) to yield a control signal that steers the crystal. One of the HP3325S is purposely set to 90kHz, so that it has a microhertz fine-tuning capability, and the other to the remaining part of the 11MHz difference frequency.

The phaselocked quartz crystal oscillator must satisfy the condition

$$(2384 + \alpha_1 + \alpha_2) \times (5 + \delta)\ \text{MHz} \simeq 11.931792\ \text{GHz} \quad (1)$$

where the 11.93GHz constant on the right hand side is the frequency of the sapphire oscillator, δ is the tuning adjustment from the broadband phaselocked loop and α_1 and α_2 are the nonintegral multiplicands provided by the ~11MHz and ~90kHz HP3325s, respectively.

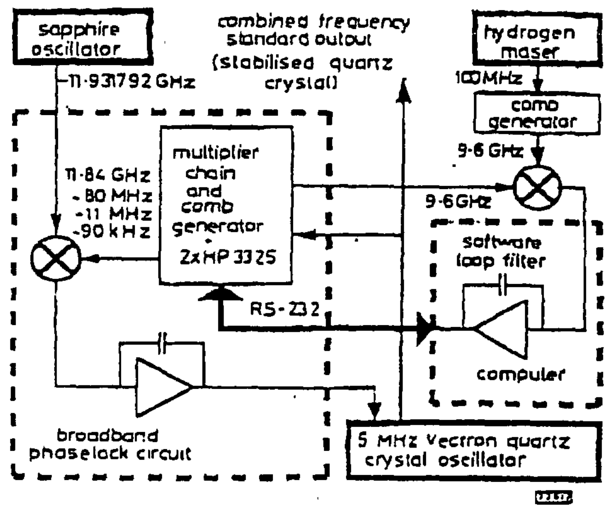


Fig. 2 *Simplified schematic diagram of the combining scheme used to stabilise a quartz crystal oscillator with a sapphire oscillator and a hydrogen maser*

Fig. 2 also shows the computer-controlled narrowband loop incorporating the hydrogen maser. To provide adequate sensitivity to the phase-difference fluctuations and to suit the available microwave hardware, the phase comparison between the crystal and the hydrogen maser was made at 9.6GHz. This tone is generated from the hydrogen maser by applying the available 100MHz signal to a step recovery diode and it is available from the crystal in the microwave comb of the broadband phaselocked loop. The beat waveform produced by mixing these tones together is digitised in the computer, processed in a software loop filter [4] and then converted to a frequency offset that is programmed into the 90 kHz HP3325. Tuning the HP3325, which provides the factor α_2 in eqn. 1, causes an overall change in the multiplicand of the broadband phaselocked loop. The broadband loop compensates for this change by adjusting the crystal offset frequency δ to maintain the phase-lock condition expressed in eqn. 1. Thus the quartz crystal is steered to follow the hydrogen maser in the long term but always remains locked to the sapphire oscillator, from which it derives its good short-term stability.

The microhertz tuning resolution of the 90kHz HP3325 permits steering of the crystal in discrete frequency steps that are much smaller than the inherent frequency fluctuations of the reference oscillators. The time constant of the software filter is set to 1000s, which is approximately the integration-time crossing point of the

C. Couvreur

Philips J. Res. 37, 231-264, 1982 R 1058

AN INTRODUCTION TO FAST GENERATION OF LARGE PRIME NUMBERS

by C. COUVREUR and J. J. QUISQUATER

Abstract

In this paper we present in the form of a survey a detailed analysis of prime distribution, sieving methods, compositeness and some primality tests. This study is aimed at adapting recent methods for generating large random primes such as needed in public-key cryptosystems. An algorithm has been implemented. It will be presented in a subsequent paper.

1. Introduction

The use of large prime numbers has revealed to be very important in modern cryptography. This has motivated an increasing interest in the subject of *prime number generation*. As an illustration let us briefly describe the *public-key cryptosystem* proposed by Rivest, Shamir and Adleman (usually referred to as the RSA or M.I.T. cryptosystem [1]). An *encryption key* consists of a pair of positive integers (e, n). The *message* M, which is an integer between 0 and $n - 1$, is encrypted into

$$C = M^e \pmod{n}.$$

Thus the cryptogram C is the remainder in the division of the e^{th} power of M by n. Similarly, a *decryption key* is a pair of positive integers (d, n). The message M is obtained by decrypting C as follows

$$M = C^d \pmod{n}.$$

The modulus n is the product of two large prime numbers, p and q. The integers e and d are multiplicative inverses modulo the least common multiple of $p - 1$ and $q - 1$, i.e.

$$e\,d \equiv 1 \pmod{\operatorname{lcm}(p-1, q-1)}.$$

The security of the RSA public-key cryptosystem is known to be crucially based on the difficulty of finding the factorization $n = pq$ of the given modulus n (see Williams [2]; Williams and Schmid [3]) and Couvreur and Goethals [4])). Some constraints must be put on p and q to create secure keys. In fact,

the process of devising suitable values for p and q requires first a method for finding large random primes between 10^{50} and 10^{100}.

Many methods for verifying that a number is a prime have been proposed. Let us now review some of them:

— *Sieve* (Eratosthenes, circa 250 b.c.) and *trial division* (Leonardo Pisano Fibonacci, 1202). These are the first known methods. Their practical limit is for numbers with a maximum of 15 to 20 digits. But combined with other tests these techniques have proved to be very powerful (see sec. 5 of the present paper).

— *Factoring*. One can test a number for primality by attempting to factorize it. Good accounts of this method are given by Guy[5]), Knuth and Pardo[6,7]), Monier[8]) and Wunderlich[9]).

— *Wilson's theorem* (published in 1770 by Waring). It characterizes the number n as being a prime if and only if the congruence $(n-1)! \equiv -1 \pmod{n}$ holds (see Hardy and Wright[10, p.68])).

— *Recognition of primes by automata*. There exists an automaton recognizing the set of primes and having a memory which grows linearly with the input length (see Hartmanis and Shank[11]).

— *Use of Pascal's arithmetic triangle*. The number n is a prime if and only if all binomial coefficients $\binom{k}{n-2k}$ are divisible by k (see Mann and Shanks[12]) and Harborth[13])).

— *Prime representing polynomials*. Certain multivariable polynomials have been explicitly determined with the following property. The set of primes is identical with the set of positive values assumed by a given polynomial as the variables range over the natural numbers (see Davis, Matijasevic and Robinson[14])).

— *Succinct prime certification*. To show that a number n is composite, it suffices to write it as a product of two nontrivial factors. There also exist analogous certificates showing that a number n is prime. However no fast way is known to find them (see Pratt[15])).

— *Exponentiation*. It is proven that there exists a certain number A greater than 1 but not an integer, such that $\lfloor A^{3^x} \rfloor$ is prime for $x = 1, 2, 3, \ldots$. Alas A is not known (see Mills[16])).

— The set of all integers which can be written as the *sum of at least three consecutive positive integers* is the set of all positive integers which are neither prime nor a power of 2 (see de la Rosa[17])).

However these methods, although interesting from a theoretical viewpoint, are not efficient for the practical problem of generating large prime numbers.

Four other basic methods for testing the primality of a number s have been developed and extensively studied. Table 1 summarizes their principal charac-

TABLE 1

Comparisons of effective methods for primality testing.

Method of primality testing	gives a rigorous proof of primality	depends on factorization	ease of implementation	speed of execution	references
1. *special functions*	yes	yes	from easy to very difficult	from slow to very rapid —	Lehmer[79,89,90]) Brillhart and al.[22]) Williams and al.[23,25]) Morrison and al.[91,92]) Adleman and Leighton[93])
2. *extension fields*	yes	no	very difficult	rapid	Adleman and Rumely[94]) Pomerance[95]) Lenstra[96,97]) Cohen[98])
3. *probabilistic (Monte-Carlo)*	no	no	easy	very rapid	Miller[66,67]) Rabin[68,69]) Solovay and Strassen[70]) Malm[99])
4. *based on the extended Riemann's hypothesis (E.R.H.)*	unknown	no	easy	rapid?	Miller[66,67]) Lenstra[100]) Vélu[101]) Mignotte[102]) Pajunen[103])

teristics (let us note that Williams[18]) presents a detailed analysis of three of these methods).

The first method requires prior knowledge of the factorization of some special functions of s, like for instance $s-1$, $s+1$ or s^2+s+1.

The second method is based on pseudoprime tests performed in various extensions of the rational numbers.

The probabilistic method is based on a simple algorithm (using k random integers out of $\{1, 2, \ldots, s-1\}$) which declares that a number s is prime with a probability of error bounded from above by 2^{-k}.

The fourth method provides certificates of primality if the Riemann hypothesis is true.

In fact the first and the second methods are the only ones to be considered when a rigorous proof of primality is required. For the first method, the problem of factoring the special function of s can be overcome by generating random factorizations instead of random numbers to be tested for primality. Williams and Schmid[3]), Buhler, Crandall and Penk[19,20]), and chiefly Plaisted[21]) propose such a method which appears to be quite general and, with some adaptations, suitable for our purposes. Thus we have adapted their method and extended their results.

The paper is organized as follows. First a large set S is constructed which consists of large odd integers s, composed from known factorizations and which are candidates for primes. The construction of the set S is described in sec. 2. Then, after discussing the distribution of primes in sec. 3, we study the average distance between two primes of S in sec. 4, thus justifying the choice of our set S. Secondly most of the composite numbers in S must be eliminated by sieving out the set S: the sieving process on the set S is studied in sec. 5. Finally the remaining elements of S must be tested for compositeness or primality; compositeness tests are analysed in sec. 6, whereas primality tests are discussed in sec. 8.

The following notations are used throughout this paper. By s we denote an integer whose primality is tested. All logarithms written in the form $\log x$ are taken with respect to the base 2. Natural logarithms are written $\ln x$.

2. Construction of the set S

The basic idea consists in constructing a set S of large odd integers s, all of which are generated using a fixed random number F, partially or completely factored. The set S must be easy to describe and must contain enough prime numbers. Some constraints are placed on F to make the compositeness or primality tests easier.

The methods we have considered for testing the primality of s use known factors of $s-1$, $s+1$, s^2-1, s^2+1, s^2+s+1 or s^2-s+1 (see Brillhart, Lehmer and Selfridge[22]); Williams and Judd[26,24]) and Williams and Holte[25])). Only the tests based on the knowledge of a complete or partial factorization of $s \pm 1$ (see Brillhart, Lehmer and Selfridge[22])) seem to be interesting here, as we do not see how to construct a family of integers s from a fixed random number F which would be a factor of s^2+1, or $s^2 \pm s - 1$. We find it useful to retain only the tests relative to the factorization of $s-1$. The tests relative to the factorization of $s+1$ have been the subject of a similar work (see Baillie and Wagstaff[104])).

In great generality, the set S to be considered here can be defined as follows:

$$S = \{s = kF + 1 \mid k = 1, 2, 3, \ldots, K\}, \tag{1}$$

where F is a random even large number. More constraints are imposed on F, depending on the compositeness or primality tests to be used in the method for generating large primes. These are discussed in the next sections. The bound K is proportional to the number of primes we want to generate and is examined in the next sections too.

3. Distribution of primes

The central theorem concerning the distribution of primes (see Hardy and Wright[10, p.9])) states that the number of primes not exceeding x, noted $\pi(x)$, is asymptotic to $x/\ln x$

$$\pi(x) \sim \frac{x}{\ln x},$$

that is

$$\lim_{x \to +\infty} \frac{\pi(x) \ln x}{x} = 1.$$

Lower and upper bounds on $\pi(x)$ have been established (see Rosser and Schoenfeld[27])), namely

$$\frac{x}{\ln x}\left(1 + \frac{1}{2\ln x}\right) < \pi(x) < \frac{x}{\ln x}\left(1 + \frac{3}{2\ln x}\right)$$

for $x \geqslant 59$, and

$$\frac{x}{\ln x - \frac{1}{2}} < \pi(x) < \frac{x}{\ln x - \frac{3}{2}}$$

for $x \geqslant 67$. It is interesting to compare the actual count of primes with the corresponding values in these formulas (see Hardy and Wright[10, p.9])); Bohman[28,29]) and Mapes[30])): these appear in table 2.

TABLE 2
Actual and approximate counts of primes

x	$\dfrac{x}{\ln x}$	$\dfrac{x}{\ln x}\left(1+\dfrac{1}{2\ln x}\right)$	$\dfrac{x}{\ln x - \frac{1}{2}}$	$\pi(x)$	$\dfrac{x}{\ln x}\left(1+\dfrac{3}{2\ln x}\right)$	$\dfrac{x}{\ln x + \frac{3}{2}}$
10^3	145	155	156	168	176	185
10^4	1 086	1 145	1 148	1 229	1 263	1 297
10^5	8 686	9 063	9 080	9 592	9 818	9 987
10^6	72 382	75 002	75 100	78 498	80 241	81 198
10^7	620 421	639 667	640 283	664 579	678 159	684 084
10^8	5 428 681	5 576 034	5 580 145	5 761 455	5 870 740	5 909 928
10^9	48 254 942	49 419 212	49 447 998	50 847 478	51 747 752	52 020 297
10^{10}	434 294 482	443 725 067	443 934 394	455 052 511	462 586 237	464 557 709
10^{11}	3 948 131 654	4 026 070 371	4 027 639 917	4 118 054 813	4 181 947 807	4 196 666 533
10^{12}	36 191 206 825	36 846 108 551	36 858 177 793	37 607 912 018	38 155 912 002	38 268 692 049
10^{13}	334 072 678 387	339 652 906 109	339 747 699 586	346 065 535 898	350 813 361 554	351 696 507 524

Other numerical expressions are known for estimating the number of primes not exceeding x. They are of an outstanding accuracy but their actual computation is not so simple as compared with the previous ones. We briefly present here the results. First the *Riemann formula* (see Knuth[6, p.366])

$$\pi(x) \sim \mu(1)\,\mathrm{Li}(x) + \mu(2)\,\frac{\mathrm{Li}(x^{\frac{1}{2}})}{2} + \mu(3)\,\frac{\mathrm{Li}(x^{\frac{1}{3}})}{3} + \ldots,$$

where $\mathrm{Li}(x)$ is the *integral logarithm* and $\mu(n)$ is the *Möbius function*. Lehmer[31]) has shown that Riemann's formula is equivalent to

$$\pi(x) \sim 1 + \frac{\ln x}{\zeta(2)} + \frac{(\ln x)^2}{2\cdot 2!\,\zeta(3)} + \frac{(\ln x)^3}{3\cdot 3!\,\zeta(4)} + \ldots,$$

where $\zeta(n)$ is the *Riemann zeta function*. Secondly the *Chebyshev formula* which was earlier conjectured by Gauss,

$$\pi(x) \sim \int_2^x \frac{\mathrm{d}x}{\ln x}.$$

In table 3 we compare the actual prime counts with those predicted by Riemann and Chebyshev (see Jones, Lal and Blundon[32]); Knuth[6, p.366]) and Shanks[33])).

In fact our interest here is in the number of primes of the form $k\,a + b$, where a and b are relatively prime. *Dirichlet's theorem* (see Hardy and Wright[10, p.13]))

TABLE 3
Actual counts of primes versus Riemann and Chebyshev counts

x	$\pi(x)$	Riemann count	Chebyshev count
10^3	168	168.36	177.6
10^4	1 229		1 246.2
10^5	9 592		9 629.6
10^6	78 498	78 527.40	78 631.7
10^7	664 579	664 667	664 918
10^8	5 761 455	5 761 551.9	5 762 208.3
10^9	50 847 478	50 847 455.4	50 849 233.9
10^{10}	455 052 511	455 050 683.3	455 055 613.5
10^{11}	4 118 054 813	4 118 052 494.6	4 118 066 399.6
10^{12}	37 607 912 018	37 607 910 542.2	37 607 950 279.8
10^{13}	346 065 535 898	346 065 531 065.8	346 065 645 809.0
10^{14}		3 204 941 731 601.7	3 204 942 065 690.9
10^{15}		29 844 570 495 886.9	29 844 571 475 286.5

asserts that there are infinitely many primes of this form. Let $\pi(x,a,b)$ denote the number of such primes which are less than or equal to x. A well-known conjecture, due to Hardy and Littlewood[34]), asserts that

$$\pi(x,a,b) \sim \frac{\pi(x)}{\phi(a)}, \tag{2}$$

where ϕ is the *Euler totient function* (this conjecture was proved by de la Vallée Poussin, for $x \to \infty$, see Apostol[35])). In table 4, we have indicated the results of a comparison between the actual value of $\pi(x,a,b)$ and its corresponding approximation (see Bays and Hudson[36-38])). As we can read from this table, $\pi(x)/\phi(a)$ is a good approximation to $\pi(x,a,b)$.

4. Average distance between two primes in the set S

Let us recall that the set S to be considered consists of numbers s of the form

$$s = kF + 1 \qquad k = 1, 2, 3, \ldots, \tag{3}$$

where F is a random even number, partially or completely factored.

Let us assume that the density of primes remains constant throughout the set S, and examine the validity of this assumption. A first approach to the question is made by considering primes of any form. Afterwards it is particularized to primes of the form (3).

We point out the experimental observation that prime numbers are randomly and uniformly distributed in a suitable length interval: indeed, as Gauss notices, if x is large while Δ is comparatively small, the number of primes between x and $x+\Delta$ is observed to be approximately given by

$$\frac{\Delta}{\ln x}$$

(see Rouse Ball and Coxeter[39])). In table 5, we examine the accuracy of this approximation on the basis of several examples taken from Mapes[30]), Jones, Lal and Blundon[32]), Zagier[40]) and Knuth[6]). We observe that, provided Δ is not too small, the approximation is generally excellent. We now examine what happens to the density of primes when a given interval is subdivided into subintervals. To that end we consider the nine successive subintervals of length 10^7 from the interval $10^7 \leqslant s \leqslant 10^8$; since there are 5 096 876 primes in $10^7 \leqslant s \leqslant 10^8$, each subinterval is expected to contain approximately 566 320 primes. In table 6 we compare the actual number of primes in these subintervals with the expected value and we observe that the ratio between the two quantities varies between 0.96 and 1.07. We find it also of interest to deter-

TABLE 4

$\pi(x,a,b)$ and Hardy-Littlewood conjecture

x	a	b	$\pi(x,a,b)$	$\phi(a)$	$\pi(x)$	$\frac{\pi(x)}{\phi(a)} \cdot \frac{1}{\pi(x,a,b)}$	gap between $\frac{\pi(x)}{\phi(a)}$ and $\pi(x,a,b)$
10^{10}	4	1	227 523 275	2	227 526 256	1.000 013 102	2 981
		3	227 529 235			0.999 986 907 2	−2 979
10^{11}	6	1	2 059 018 668	2	2 059 027 407	1.000 004 244	8 739
		5	2 059 036 143			0.999 995 757 2	−8 736
10^{11}	24	1	514 742 404	8	514 756 852	1.000 028 068	14 448
		5	514 760 074			0.999 993 740 8	−3 222
		7	514 762 733			0.999 988 575 3	−5 881
		11	514 755 092			1.000 003 419	1 760
		13	514 757 222			0.000 000 001 9	− 370
		17	514 760 580			0.999 992 757 8	−3 728
		19	514 756 309			1.000 001 055	543
		23	514 760 397			0.999 993 113 3	−3 545
10^{12}	3	1	18 803 933 520	2	18 803 956 010	1.000 001 196	22 490
		2	18 803 978 497			0.999 998 804 5	−22 487
10^{12}	24	1	4 700 968 265	8	4 700 989 001	1.000 004 411	20 736
		5	4 700 998 678			0.999 997 941 5	−9 677
		7	4 701 001 698			0.999 997 299 1	−12 697
		11	4 701 004 728			0.999 996 654 5	−15 727
		13	4 700 973 812			1.000 003 231	15 189
		17	4 700 983 585			1.000 001 152	5 416
		19	4 700 989 745			0.999 999 841 7	− 744
		23	4 700 991 505			0.999 999 467 3	−2 504

TABLE 5
Prime counts in given intervals

interval $[x, x+\Delta]$	$\pi(x \leqslant s \leqslant x+\Delta)$	$\frac{\Delta}{\ln x}$	$\frac{\pi(x \leqslant s \leqslant s+\Delta)}{\Delta/\ln x}$
$10^7 \leqslant s \leqslant 10^9$	50 182 955	61 421 648	0.8170
$10^8 \leqslant s \leqslant 10^9$	45 086 079	48 858 129	0.9228
$10^7 \leqslant s \leqslant 2*10^7$	606 028	620 421	0.9768
$2*10^7 \leqslant s \leqslant 3*10^7$	587 252	594 840	0.9872
$3*10^7 \leqslant s \leqslant 4*10^7$	575 795	580 831	0.9913
$4*10^7 \leqslant s \leqslant 5*10^7$	567 480	571 285	0.9933
$5*10^7 \leqslant s \leqslant 6*10^7$	560 981	564 094	0.9945
$6*10^7 \leqslant s \leqslant 7*10^7$	555 949	558 352	0.9957
$7*10^7 \leqslant s \leqslant 8*10^7$	551 318	553 587	0.9959
$8*10^7 \leqslant s \leqslant 9*10^7$	547 572	549 525	0.9964
$9*10^7 \leqslant s \leqslant 10^8$	544 501	545 991	0.9973
$10^8 \leqslant s \leqslant 10^8 + 150\,000$	8 154	8 143	1.0013
$10^7 \leqslant s \leqslant 10^7 + 100$	2	6	0.3224
$2^{24} - 167 \leqslant s \leqslant 2^{24} - 3$	10	9.9	1.0144
$2^{25} - 183 \leqslant s \leqslant 2^{25} - 39$	10	8.3	1.2034
$2^{26} - 135 \leqslant s \leqslant 2^{26} - 5$	10	7.2	1.3863
$2^{27} - 235 \leqslant s \leqslant 2^{27} - 39$	10	10.5	0.9548
$2^{28} - 273 \leqslant s \leqslant 2^{28} - 57$	10	11.1	0.8985
$2^{29} - 133 \leqslant s \leqslant 2^{29} - 3$	10	6.5	1.5463
$10^8 - 213 \leqslant s \leqslant 10^8 - 11$	10	11.0	0.9119
$10^9 - 267 \leqslant s \leqslant 10^9 - 63$	10	9.8	1.0158

TABLE 6
Variation of density of primes

intervals	actual number of primes	expected number of primes	ratio between the actual number of primes and the expected one
$10^7 \leqslant s \leqslant 10^8$	5 096 876		
$10^7 \leqslant s \leqslant 2*10^7$	606 028	566 320	1.0701
$2*10^7 \leqslant s \leqslant 3*10^7$	587 252	566 320	1.0370
$3*10^7 \leqslant s \leqslant 4*10^7$	575 795	566 320	1.0167
$4*10^7 \leqslant s \leqslant 5*10^7$	567 480	566 320	1.0020
$5*10^7 \leqslant s \leqslant 6*10^7$	560 981	566 320	0.9906
$6*10^7 \leqslant s \leqslant 7*10^7$	555 949	566 320	0.9817
$7*10^7 \leqslant s \leqslant 8*10^7$	551 318	566 320	0.9735
$8*10^7 \leqslant s \leqslant 9*10^7$	547 572	566 320	0.9669
$9*10^7 \leqslant s \leqslant 10^8$	544 501	566 320	0.9615

mine the average and maximum differences between two consecutive primes in a given interval $[x, x+\Delta]$. The following formula due to Cadwell[41]) gives the mean value for the largest gap

$$2 + (\ln x - 2)\left(\ln \frac{\Delta}{\ln x} + \gamma\right),$$

where γ is *Euler's constant* ($\gamma = 0.57721\ldots$, see sec. 5). The average gap is of course $\ln x$. Some numerical results are given in table 7 (see Cadwell[41]); Knuth[6]); Zagier[40]) and Weintraub[42-44])). They allow us to verify how accurately the largest gap may be estimated and to compare the average gap with the largest one.

We are now in a position to particularize this approach to primes of the form $kF+1$. Considering that they are also randomly and uniformly distributed in a suitable length interval, we may estimate that the number of such primes between x and $x+\Delta$ is given by

$$\frac{\Delta}{\phi(F)\ln x}. \tag{4}$$

We have estimated the accuracy of this formula on several examples, on the basis of data from Bays and Hudson[36]) and Shanks[33]). The numerical results are reported in table 8 and allow us to conclude that (4) is a slightly overestimating, but generally excellent approximation. Let us now examine the validity of the assumption that the density of such primes remains constant throughout a considered interval. An analysis similar to that made for primes of any form has been conducted. The results are given in table 9. We consider successive subintervals from several given intervals and we compare the actual number of primes in these subintervals with the expected value. As far as the chosen examples are concerned, the ratio between the two counts varies between 0.97 and 1.05.

The preceding discussion allows us to conclude that, in the interval $[x, x+\Delta]$, the average distance between two consecutive primes of the form $kF+1$ may be roughly estimated by $\phi(F)\ln x$. Hence it seems that we may estimate that the number of such primes in the set S is in a ratio of one out $\ln x\,\phi(F)/F$ elements of S.

Let us now have a closer look to the ratio $\phi(F)/F$. Let $F = p_1^{e_1} p_2^{e_2} \ldots p_n^{e_n}$ denote the complete factorization of F; then

$$\phi(F) = F\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\ldots\left(1 - \frac{1}{p_n}\right).$$

TABLE 7
Gaps between primes

interval $[x, x+\Delta]$	observed maximum gap	calculated maximum gap	ratio between the observed maximum gap and the calculated one	average gap
$10^8 \leqslant s \leqslant 10^8 + 150\,000$	168	159	1.0543	18
$10^9 \leqslant s \leqslant 10^9 + 150\,000$	176	179	0.9821	21
$10^{10} \leqslant s \leqslant 10^{10} + 150\,000$	182	199	0.9156	23
[illegible] $\leqslant s \leqslant 10^{11} + 150\,000$	148	218	0.6786	25
$10^{12} \leqslant s \leqslant 10^{12} + 150\,000$	222	237	0.9359	28
$10^{13} \leqslant s \leqslant 10^{13} + 150\,000$	234	256	0.9137	30
$10^{14} \leqslant s \leqslant 10^{14} + 150\,000$	300	275	1.0917	32
$10^{15} \leqslant s \leqslant 10^{15} + 150\,000$	276	293	0.9409	35
$10^3 \leqslant s \leqslant 10^4$	36	40	0.8993	7
$10^4 \leqslant s \leqslant 10^5$	72	72.4	0.9944	9
$10^5 \leqslant s \leqslant 10^6$	114	115	0.9942	12
$10^6 \leqslant s \leqslant 10^7$	154	167	0.9222	14
$10^7 \leqslant s \leqslant 10^8$	220	229	0.9587	16
$10^8 \leqslant s \leqslant 10^9$	292	302	0.9332	18
$10^9 \leqslant s \leqslant 10^{10}$	354	385	0.9190	21
$2^{24} - 167 \leqslant s \leqslant 2^{24} - 3$	50	44	1.1380	17
$2^{25} - 183 \leqslant s \leqslant 2^{25} - 39$	26	43	0.6004	17
$2^{26} - 135 \leqslant s \leqslant 2^{26} - 5$	42	43	0.9789	18
$2^{27} - 235 \leqslant s \leqslant 2^{27} - 39$	52	51	1.0215	19
$2^{28} - 273 \leqslant s \leqslant 2^{28} - 57$	60	54	1.1112	19
$2^{29} - 133 \leqslant s \leqslant 2^{29} - 3$	30	46	0.6488	20
$10^8 - 213 \leqslant s \leqslant 10^8 - 11$	84	51	1.6535	18
[illegible] $- 267 \leqslant s \leqslant 10^9 - 63$	86	56	1.5461	21
[illegible] $- 231 \leqslant s \leqslant 10^{10} - 33$	48	59	0.8084	23
$10^{11} - 231 \leqslant s \leqslant 10^{11} - 23$	52	65	0.8051	25
$10^7 - 100 \leqslant s \leqslant 10^7 + 100$	60	46	1.3128	16
$10^{14} \leqslant s \leqslant 10^{14} + 10^8$	414	471	0.8782	32
$10^{17} \leqslant s \leqslant 10^{17} + 10^7$	432	486	0.8890	39
$1.1 \cdot 10^{16} \leqslant s \leqslant 1.1 \cdot 10^{16} + 10^8$	546	540	0.9883	37
$1.1 \cdot 10^{16} + 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 2 \cdot 10^8$	468	540	1.1531	37
$1.1 \cdot 10^{16} + 2 \cdot 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 3 \cdot 10^8$	484	540	1.1149	37
$1.1 \cdot 10^{16} + 3 \cdot 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 4 \cdot 10^8$	510	540	1.0581	37
$1.1 \cdot 10^{16} + 4 \cdot 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 5 \cdot 10^8$	528	540	1.0220	37
$1.1 \cdot 10^{16} + 5 \cdot 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 6 \cdot 10^8$	504	540	1.0707	37
$1.1 \cdot 10^{16} + 6 \cdot 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 7 \cdot 10^8$	494	540	1.0924	37
$1.1 \cdot 10^{16} + 7 \cdot 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 8 \cdot 10^8$	484	540	1.1149	37
$1.1 \cdot 10^{16} + 8 \cdot 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 9 \cdot 10^8$	460	540	1.1731	37
$1.1 \cdot 10^{16} + 9 \cdot 10^8 \leqslant s \leqslant 1.1 \cdot 10^{16} + 10^9$	486	540	1.1103	37

TABLE 8
Prime counts in given intervals and forms

interval $[x, x+\Delta]$	$\pi(x \leqslant s \leqslant x+\Delta, 24, 1)$	$\dfrac{\Delta}{8 \cdot \ln x}$	$\dfrac{\pi(x \leqslant s \leqslant x+\Delta, 24, 1)}{\Delta/(8 \cdot \ln x)}$
$10^{11} \leqslant s \leqslant 2 \cdot 10^{11}$	486 130 233	493 516 457	0.9850
$2 \cdot 10^{11} \leqslant s \leqslant 3 \cdot 10^{11}$	476 410 254	480 370 464	0.9918
$3 \cdot 10^{11} \leqslant s \leqslant 4 \cdot 10^{11}$	470 320 943	473 000 233	0.9943
$4 \cdot 10^{11} \leqslant s \leqslant 5 \cdot 10^{11}$	465 887 425	467 906 650	0.9957
$5 \cdot 10^{11} \leqslant s \leqslant 6 \cdot 10^{11}$	462 420 065	464 030 681	0.9965
$6 \cdot 10^{11} \leqslant s \leqslant 7 \cdot 10^{11}$	459 568 492	460 911 132	0.9971
$7 \cdot 10^{11} \leqslant s \leqslant 8 \cdot 10^{11}$	457 165 032	458 306 128	0.9975
$8 \cdot 10^{11} \leqslant s \leqslant 9 \cdot 10^{11}$	455 082 970	456 073 257	0.9978
$9 \cdot 10^{11} \leqslant s \leqslant 10^{12}$	453 240 447	454 121 708	0.9981
	$\pi(x \leqslant s \leqslant x+\Delta, 8, 1)$	$\dfrac{\Delta}{4 \cdot \ln x}$	$\dfrac{\pi(x \leqslant s \leqslant x+\Delta, 8, 1)}{\Delta/(4 \cdot \ln x)}$
$250\,000 \leqslant s \leqslant 500\,000$	4861	5028	0.9667
$500\,000 \leqslant s \leqslant 750\,000$	4664	4763	0.9792
$750\,000 \leqslant s \leqslant 1\,000\,000$	4554	4620	0.9857
	$\pi(x \leqslant s \leqslant x+\Delta, 10, 1)$	$\dfrac{\Delta}{4 \cdot \ln x}$	$\dfrac{\pi(x \leqslant s \leqslant x+\Delta, 10, 1)}{\Delta/(4 \cdot \ln x)}$
$250\,000 \leqslant s \leqslant 500\,000$	4891	5028	0.9727
$500\,000 \leqslant s \leqslant 750\,000$	4641	4763	0.9744
$750\,000 \leqslant s \leqslant 1\,000\,000$	4590	4620	0.9935
	$\pi(x \leqslant s \leqslant x+\Delta, 12, 1)$	$\dfrac{\Delta}{4 \cdot \ln x}$	$\dfrac{\pi(x \leqslant s \leqslant x+\Delta, 12, 1)}{\Delta/(4 \cdot \ln x)}$
$252\,000 \leqslant s \leqslant 504\,000$	4900	5065	0.9673
$504\,000 \leqslant s \leqslant 756\,000$	4696	4798	0.9787
$756\,000 \leqslant s \leqslant 1\,008\,000$	4615	4654	0.9916

Thus we may conclude that the smaller $\phi(F)/F$ is (that is, the more small prime factors F has), the more prime numbers S contains. In any case, as F is an even integer, $\phi(F)/F$ is always less than 1/2. More precisely the average value of $\phi(F)/F$ is $3/\pi^2 \sim 0.30396$ (see Apostol[36, p.62]).

5. Sieves

A *sieve* is a combinatorial technique that allows one to eliminate all the unwanted members of a finite set by a finite sequence of well-defined steps (for the history of the sieve process, see Lehmer[45]).

TABLE 9

Variations of density of primes of the form $kF+1$

interval $[x, x+\Delta]$	$\pi(x \leqslant s \leqslant x+\Delta,24,1)$	expected number of primes	ratio between the actual count and the expected one
$10^{11} \leqslant s \leqslant 10^{12}$	4 186 225 861	465 136 207 (= 4 186 225 861/9)	
$10^{11} \leqslant s \leqslant 2 \cdot 10^{11}$	486 130 233		1.0451
$2 \cdot 10^{11} \leqslant s \leqslant 3 \cdot 10^{11}$	476 410 254	,,	1.0242
$3 \cdot 10^{11} \leqslant s \leqslant 4 \cdot 10^{11}$	470 320 943	,,	1.0111
$0^{11} \leqslant s \leqslant 5 \cdot 10^{11}$	465 887 425	,,	1.0016
$5 \cdot 10^{11} \leqslant s \leqslant 6 \cdot 10^{11}$	462 420 065	,,	0.9942
$6 \cdot 10^{11} \leqslant s \leqslant 7 \cdot 10^{11}$	459 568 492	,,	0.9880
$7 \cdot 10^{11} \leqslant s \leqslant 8 \cdot 10^{11}$	457 165 032	,,	0.9829
$8 \cdot 10^{11} \leqslant s \leqslant 9 \cdot 10^{11}$	455 082 970	,,	0.9784
$9 \cdot 10^{11} \leqslant s \leqslant 10^{12}$	453 240 447	,,	0.9744
	$\pi(x \leqslant s \leqslant x+\Delta,8,1)$		
$250\,000 \leqslant s \leqslant 1\,000\,000$	14 079		
$250\,000 \leqslant s \leqslant 500\,000$	4 861	4 693	1.0358
$500\,000 \leqslant s \leqslant 750\,000$	4 664	,,	0.9938
$750\,000 \leqslant s \leqslant 1\,000\,000$	4 554	,,	0.9704
	$\pi(x \leqslant s \leqslant x+\Delta,10,1)$		
$250\,000 \leqslant s \leqslant 1\,000\,000$	14 122		
$250\,000 \leqslant s \leqslant 500\,000$	4 891	4 707	1.0390
$500\,000 \leqslant s \leqslant 750\,000$	4 641	,,	0.9859
$00 \leqslant s \leqslant 1\,000\,000$	4 590	,,	0.9751
	$\pi(x \leqslant s \leqslant x+\Delta,12,1)$		
$252\,000 \leqslant s \leqslant 1\,008\,000$	14 211		
$252\,000 \leqslant s \leqslant 504\,000$	4 900	4 737	1.0344
$504\,000 \leqslant s \leqslant 756\,000$	4 696	,,	0.9913
$756\,000 \leqslant s \leqslant 1\,008\,000$	4 615	,,	0.9742

The sieve idea plays a fundamental role in the theory of numbers. Deshouillers[46]), Halberstam and Richert[47]) and Hooley[48]) describe the theoretical frame for the sieves. Here we shall only give an elementary account of sieve methods which are relevant to the problem of prime generation.

The first known method for determining primes is the *sieve of Eratosthenes*. It eliminates the composite numbers between $n^{\frac{1}{2}}$ and n in the following way:

all multiples of the first prime, i.e. 2, are removed; then all multiples of the next prime, i.e. 3, are removed, and so on. The process stops after sifting with the largest prime less than $n^{\frac{1}{2}}$.

The sieve of Eratosthenes is computationally fast and easily implemented since the multiples of a number may be computed by successive additions or shifts. Let p_i be the ith prime. Then the number of "cross-out" operations, i.e. essentially the computation time, is given by

$$t = n\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} \cdots \frac{1}{p_k}\right),$$

where p_k is the largest prime not exceeding $n^{\frac{1}{2}}$. By use of

$$\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \ldots + \frac{1}{p} \sim \int_2^p \frac{\mathrm{d}p}{p \ln p} \sim \ln \ln p, \tag{5}$$

we obtain

$$t = \sum_{i=1}^{\pi(n^{\frac{1}{2}})} \frac{1}{p_i} \sim n \ln \ln n^{\frac{1}{2}}.$$

We conclude that, for any practical purpose, the computation time is essentially linear with respect to n (for instance, for $n = 10^{10}$, $\ln \ln 10^5 \sim 2.4435$).

Nearly all prime number generators use the sieving principle. Elementary algorithms are described by Chartres[49]), Singleton[50]) and Wood[51]). The complexity of sieve processes is studied by Mairson[52]) and Gries and Misra[53]). These authors give an algorithm of arithmetic complexity $0(n)$ and show that, under the RAM model of computation, this upper bound is equivalent to the theoretical lower bound. Another algorithm, the arithmetic complexity of which is only $0(n/\ln \ln n)$, is presented by Pritchard[54]). Bays and Hudson[36]) show how the problem of the large amount of memory required can be removed. Wells[55]) and Wunderlich[56]) give the most efficient methods for representing sets and discuss the programming difficulties encountered when sieving out these sets.

All published tables are computed by application of the sieve of Eratosthenes. Large or compact lists of primes may be found in Lehmer[31]), Weintraub[57]) and Baker and Gruenberger[58]), up to the prime 104 395 289. Special devices for sieving are announced or described by Lehmer[59,60]) and Cantor, Estrin, Fraenkel and Turn[61]) with rates of sieving up to 10^{10} numbers/min. Finally, let us remark that the most efficient sieves of Eratosthenes permit to generate primes up to about 10^{16}, which is much too low for cryptographic applica-

tions. However, these sieves may prove useful in the preliminary generation of factors in a random factorization.

The *sieving set with sieve limit* x, denoted by P_x, is defined to be the set of all primes p with $2 \leqslant p \leqslant x$. Then, the function $Q(x)$ is defined by

$$Q(x) = \prod_{p \in P_x} \left(1 - \frac{1}{p}\right).$$

For an integer n much larger than x but otherwise random, we loosely interpret $Q(x)$ as the probability for n to be relatively prime to all primes in P_x.

:rom the approximation

$$e^{-1/p} \sim 1 - \frac{1}{p},$$

we find

$$Q(x) \sim \exp\left(-\sum_p \in p_x \frac{1}{p}\right),$$

and from (5) we conclude that

$$Q(x) \sim e^{-\ln \ln x} \sim \frac{1}{\ln x}.$$

In fact, the *Mertens theorem* (see Hardy and Wright[10, p.351])) gives the estimation

$$Q(x) \sim \frac{e^{-\gamma}}{\ln x} \sim \frac{0.56145948\ldots}{\ln x}, \tag{6}$$

where γ is the Euler or Mascheroni constant. This constant is defined by

$$\gamma = \lim_{n \to +\infty} \left(1 + \frac{1}{2} + \ldots + \frac{1}{n} - \ln n\right)$$

and is known up to at least 30 100 decimal places (see Brent and McMillan[62])). The value $\gamma \sim 0.577215664902$ is sufficient for our purposes.

Table 10 displays the actual function $Q(x)$ and its estimate (6) for some values of s. Further numerical data can be found in Appel and Rosser[63]).

If a set P_x is used to sieve out the set of odd numbers

$$S = \{s = kF + 1 \mid 1 \leqslant k \leqslant K\},$$

as defined in (1), the function $2Q(x)$ gives the average ratio of surviving numbers.

The problem of determining the most efficient sieve limit x is thus reduced to a straightforward minimization problem involving the relevant asymptotic formulas (see Crandall and Penk[20]) for an example).

TABLE 10
The function Q and its estimation

x	greatest prime $< x = p$	$Q(p)$	$\frac{e^{-\gamma}}{\ln p}$
10	7	0.228 571 558 192	0.288 533 097 9
10^2	97	0.120 317 304 818	0.122 731 137 8
10^3	997	0.080 965 273 159	0.081 314 952 89
10^4	9 973	0.060 884 699 714	0.060 977 588 56
10^5	99 991	0.048 752 923 663	0.048 768 132 36
5.10^5	499 979	0.042 781 472 584	0.042 786 597 53
10^6	999 983	0.040 638 215 016	0.040 639 842 58

If a set P_x is used to sieve out a small set $S = \{s = kF + 1 \mid 1 \leqslant k \leqslant K\}$ of large numbers, the classical techniques of sieving are prohibitive. More relevant techniques are:

— successive divisions by the numbers from P_x,
— computation of $\gcd(s, N_x)$, for each $s \in S$, where N_x is the product of first primes up to x. If $\gcd(s, N_x) \neq 1$, s is eliminated from S. If x is too large, it might be necessary to compute several gcd's. If N_x is a number larger than s, the computation of this gcd requires at most $2.078 \ln N_x$ divisions using the Euclidean algorithms (see Knuth[6, p.343])). Other efficient algorithms exist for computing gcd's without any division (which is a relatively slow operation).

We compute now the function $\ln N_x$. Let us put Chebyshev's θ-function (see Apostol[35, p.75])) defined by

$$\theta(x) = \ln \prod_{p \leqslant x} p.$$

A good approximation to $\theta(x)$ is given by

$$\theta(x) = \sum_{p \leqslant x} \ln p \leqslant \pi(x) \ln x,$$

i.e., from the prime number theorem,

$$\theta(x) \sim x.$$

Among other results about the function θ, Schoenfeld[64]) shows, by extensive computations, that

$$x - 2.05x^{\frac{1}{2}} < \theta(x) < x,$$

for $0 \leqslant x \leqslant 10^{11}$. Table 11 gives some values of $\theta(x)$. For $x = 500\,000$, our computation confirms the value given by Johnson[65]).

In conclusion this section has shown that the partial sieving is fruitful even for large numbers. *A good value for the sieving limit x seems to be about* 1 000.

TABLE 11
The product of first primes and its estimation

x	p	$\theta(p)$	$p - 2.05\,p^{1/2}$
10	7	5.347 107 531	1.576 209 812
100	97	83.728 390 399	76.809 841 51
1 000	997	956.245 265 12	932.270 621 0
10 000	9 973	9 895.991 379 2	9 768.276 937
100 000	99 991	99 685.389 2	99 342.762 25
500 000	499 979	499 318.120	498 529.461 5
1 000 000	999 983	998 484.175	997 933.017 4

6. Compositeness tests

Miller[66,67]), Rabin[68,69]) and Solovay and Strassen[70]) have developed fast stochastic methods to recognize the compositeness of a number. These methods make no use of any factorization and allow one to build efficient *probabilistic* primality testing algorithms.

In this section, we first discuss some extensions of Fermat's theorem and other number-theoretic concepts which are needed for the presentation of these tests. We refer to Baratz[71]), Chaitin and Schwartz[72]), Monier[73,74]) and Selfridge (unpublished) for an analysis of their performance. Then, as applications, we present exact and probabilistic primality tests and compositeness tests. Our treatment is closely related to Monier's discussion and to Selfridge's ideas presented by Williams[18]).

Let $s = p_1^{e_1} p_2^{e_2} \ldots p_n^{e_n}$ be the *prime decomposition* of s, and let us define the following functions:

(i) The *Carmichael function* $\lambda(s) = \operatorname{lcm}(\phi(p_n^{e_n}))$, also called the *indicator* of s.

(ii) The *dyadic valuation* $v_2(s) = \max\{k \text{ such that } 2^k | s\}$.

(iii) The *Jacobi symbol* $(\frac{a}{s})$, for an odd integer s and any integer a, where $(\frac{a}{p})$ is the *Legendre symbol*, i.e., the integer in the set $\{0, 1, -1\}$ congruent to $a^{(p-1)/2}$ modulo p (note that $(\frac{a}{s}) = 0$ if and only if $\gcd(a,s) \neq 1$).

Let s be an odd prime power. Then the set of positive integers less than s and relatively prime with s is known to be of the form $\{g^i (\operatorname{mod} s) | 0 \leqslant i \leqslant \phi(s) - 1\}$, for a suitable integer g called a *primitive root* of s. Thus for each a with $\gcd(a,s) = 1$ there exists a unique exponent i in the interval $0 \leqslant i \leqslant \phi(s) - 1$ such that $g^i \equiv a \pmod{s}$; this exponent is called the *index* of a modulo s (relative to g) and is denoted by $\operatorname{ind}(a)$. The theory of indices bears certain similarities with that of logarithms, the primitive root g playing the role of the base.

The least positive integer t such that $a^t \equiv 1 (\operatorname{mod} s)$ is called the *order* of a in the group of reduced residues $(\operatorname{mod} s)$ and is denoted by $\operatorname{ord}_s(a)$.

Let us now recall Fermat's theorem.

Theorem 1. (Fermat, 1640: see Apostol[35, p.114]). *If s is prime and* $\gcd(a,s) = 1$, then

$$a^{s-1} \equiv 1 \pmod{s}. \tag{7}$$

■

Therefore if $a^{s-1} \not\equiv 1 (\operatorname{mod} s)$ for some a with $1 < a < s$, then s must be composite. It is not true that if (7) holds for some a, then s is prime. For example, consider the composite number $341 = 11 \cdot 31$. As $2^{10} - 1 = 3 \cdot 11 \cdot 31$, we have $2^{10} \equiv 1 (\operatorname{mod} 341)$, hence $2^{340} \equiv 1 (\operatorname{mod} 341)$. We call any integer s satisfying (7) for a given a a "*base a-pseudo-prime*" or *a-psp*.

Theorem 2. (Cipolla[75]): see Williams[18, p.139]) and Hardy and Wright[10, p.72])). *For each $a > 1$, there exist infinitely many composite a-psp's.*

Proof.

Let p be an odd prime such that $\gcd(p, a^2 - 1) = 1$. The number

$$s = \frac{a^{2p} - 1}{a^2 - 1} = \frac{a^p - 1}{a - 1}\,\frac{a^p + 1}{a + 1}$$

is clearly a composite number. We have

$$a^{2p} \equiv s(a^2 - 1) + 1 \equiv 1 \pmod{s},$$

and

$$s - 1 = \left(\frac{a^{p-1} - 1}{a^2 - 1}\right)(a^{p-1} + 1)\,a^2.$$

Since $a^{p-1} - 1$ is divisible by $p(a^2 - 1)$ and $(a^{p-1} + 1)a^2$ is always an even integer, we conclude that $s - 1$ is divisible by $2p$, whence

$$a^{s-1} \equiv 1 \pmod{s}.$$

■

Notice that, if (7) is satisfied for all a relatively prime to s, then s is not necessarily prime. There exist composite numbers having that property. They are called *Carmichael numbers*. These numbers are characterized by Carmichael[23]) who obtains the following result.

Theorem 3. *The necessary and sufficient condition for a number s to be a Carmichael number is that $\lambda(s) | s - 1$. A Carmichael number is odd and equal to a product of distinct primes $s = p_1 p_2 \ldots p_n$, with $n \geqslant 3$.* ■

The smallest Carmichael numbers are $561 = 3 \cdot 11 \cdot 17$, $1105 = 5 \cdot 13 \cdot 17$ and $1729 = 7 \cdot 13 \cdot 19$ (for $a = 2$, these numbers are not given by Cipolla's construction). Methods for constructing Carmichael numbers are devised by Chernick[76]) and Sispanov[77]). A detailed account of these methods is given by Ore[78, p. 331–339]). It is still not known whether infinitely many Carmichael numbers exist or not.

We denote by $CP_2(x)$ the number of composite 2-psp's less than or equal to x and by $C(x)$ the number of Carmichael numbers less than or equal to x. Tables of numerical values for these numbers are constructed by Lehmer[79]), [illegible]let[80]), Swift[81]), Monier[73]) (this last table is incomplete) and Pomerance, Selfridge and Wagstaff[82,83]). In table 12 we summarize these results, which suggest the following asymptotic relations:

$$CP_2(x) \sim \pi(x)^{0.482},$$

and

$$C(x) \sim 0.155\, x^{0.4}.$$

Table 12 shows in particular that, for $x = 10^{10}$, the number of primes satisfying (7) for $a = 2$ is 30 571 times larger than the number of non-primes with the same property.

In order to handle the Carmichael numbers, some refinement of Fermat's theorem and some new concepts are to be introduced.

TABLE 12

Composite 2-psp's and Carmichael numbers

x	$\pi(x)$	$CP_2(x)$	$\pi(x)^{0.482}$	$C(x)$	$0.155\,x^{0.4}$	$\dfrac{CP_2(x)}{C(x)}$
10^3	168	3	12	1	2	3
10^4	1 229	22	31	7	6	3.143
10^5	9 592	78	83	16	16	4.875
10^6	78 498	245	229	43	39	5.698
10^7	664 579	750	640	105	98	7.143
10^8	5 761 455	2 057	1 814	255	246	8.067
10^9	50 847 478	5 597	5 181	646	617	8.664
10^{10}	455 052 511	14 884	14 900	1 547	1 550	9.622
$25 \cdot 10^9$		21 853		2 163	2 236	10.103

An "*Euler base a-pseudoprime*" is an odd integer s such that

$$a^{(s-1)/2} \equiv \left(\frac{a}{s}\right) \pmod s.$$

If s is an odd prime and $\gcd(a,s) = 1$, then s is an *Euler a-psp*.

Following Selfridge and Williams[18]), we define a "*strong base a-pseudoprime*" to be an odd integer s such that, writing $s - 1 = 2^{\nu_0}s'$, s' odd, one has either

$$a^{s'} \equiv 1 \pmod s$$

or

$$a^{2^k s'} \equiv -1 \pmod s,$$

for some k, $0 < k < \nu_0$. Again, if s is an odd prime and $\gcd(a,s) = 1$, then s is a *strong a-psp*.

An odd integer s satisfies the *property MR* (see Miller[66,67])) for a given a if

$$a^{s-1} \equiv 1 \pmod s$$

and for every integer k, $0 < k \leqslant \nu_0$,

$$a^{(s-1)/2^k} \not\equiv 1 \pmod s$$

implies

$$\gcd(a^{(s-1)/2^k} - 1, s) = 1.$$

Theorem 4. (Selfridge, see Williams[18])). *An odd integer s satisfies the property MR for a given a if and only if it is a strong a-psp.*

Proof.

First, suppose s satisfies the property MR for a given a.

— If no value of k exists such that $a^{(s-1)/2^k} \not\equiv 1 \pmod s$, then $a^{(s-1)/2^k} \equiv 1$ for all possible k's and thus $a^{s'} \equiv 1 \pmod s$.

— Otherwise, let k be the least integer such that $a^{(s-1)/2^k} \not\equiv 1 \pmod s$. Then

$$a^{(s-1)/2^k} \equiv 1 \pmod s,$$

hence

$$(a^{(s-1)/2^k} - 1)(a^{(s-1)/2^k} + 1) \equiv 0 \pmod s.$$

From $\gcd(s, a^{(s-1)/2^k} - 1) = 1$, we conclude that

$$a^{(s-1)/2^k} = a^{2^{\nu_0-k}s'} \equiv -1 \pmod s$$

and thus s is a strong a-psp.

Conversely, suppose now that s is a strong a-psp.

— If $a^{s'} \equiv 1 \pmod s$, then $a^{2^i s'} \equiv 1 \pmod s$, $0 \leqslant i \leqslant \nu_0$.

— If $a^{2^k s'} \equiv -1 \pmod s$ for some k, $0 \leqslant k \leqslant \nu_0$, then $a^{2^{k+1}s'} \equiv 1 \pmod s$.

Hence $\nu_0 - k$ is the value of the least positive integer l such that $a^{(s-1)/2^l} \not\equiv 1 \pmod s$. From

$$a^{(s-1)/2^{\nu_0-k}} + 1 \equiv 0 \pmod s,$$

we conclude that

$$\gcd(a^{(s-1)/2^{\nu_0-k}} - 1, s) = 1,$$

i.e. satisfies the property MR for a. ■

We now require the following result.

Theorem 5. see Pocklington[84]). *Let q be any prime such that $q^e | m$ and $q^{e+1} \nmid m$. If $a^m \equiv 1 \pmod s$ and $\gcd(a^{m/q} - 1, s) = 1$, then any prime factor of must have the form $1 + kq^e$.* ■

This result produces the following theorem proved by Selfridge[18]) and Monier[73,74]).

Theorem 6. *If s is a strong a-psp, then s is an Euler a-psp.*

Sketched proof.

— First, if $a^{s'} \equiv 1 \pmod s$, then by Fermat's theorem and since s' is odd, $a^{(p-1)/2} \equiv 1 \pmod s$ for each prime divisor p of s. From this, we conclude that

$$a^{(s-1)/2} \equiv \left(\frac{a}{s}\right) \pmod s.$$

— Otherwise if $a^{2^r s'} \equiv -1 \pmod s$ for some r with $0 \leqslant r < \nu_0$, we have by theorem 5 that $d | 2^{r+1}s'$ and $d \nmid 2^r s'$ where $d = \mathrm{ord}_p(a)$ and p is any prime divisor of s. Thus d is an odd multiple of 2^{r+1} and $p = 2^{r+1}k + 1$.

By definition,

$$a^{d/2} \equiv -1 \pmod p$$

·d

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \equiv (-1)^{(p-1)/d} \pmod p.$$

Thus

$$\left(\frac{a}{p}\right) \equiv (-1)^k.$$

From the prime decomposition $s = \prod_{i=1}^n p_i^{e_i}$ and from $p_i = 2^{r+1}k_i + 1$, we have

$$s \equiv 1 + 2^{r+1} \sum_{i=1}^n e_i k_i \pmod{2^{2r+2}}.$$

Thus

$$s' 2^{\nu_0 - 1} = (s-1)/2 \equiv 2^r \sum_{i=1}^n e_i k_i \pmod{2^{r+1}}$$

and

$$2^{\nu_0 - 1 - r} \equiv \sum_{i=1}^n e_i k_i \pmod 2.$$

The theorem follows from

$$a^{(s-1)/2} \equiv (-1)^{2^{\nu_0-1-r}} = (-1)^{\sum_{i=1}^n e_i k_i} \pmod s$$

and

$$\left(\frac{a}{s}\right) = \prod_{i=1}^n \left(\frac{a}{p_i}\right)^{e_i} = (-1)^{\sum_{i=1}^n e_i k_i}.$$

■

For any odd composite numbers s, let $L_{MR}(s)$ denote the set of all a, $1 \leqslant a < s$, such that s is a strong a-psp. For computing the size of $L_{MR}(s)$, we need to know the number of solutions to the equation $x^t \equiv b \pmod s$ in the unknown x.

Theorem 7. (see Monier[74]). *Let s be an odd integer, $s = p_1^{e_1} \ldots p_n^{e_n}$ its prime factorization, s an integer such that $\gcd(b, s) = 1$, t a positive integer and $d_i = \mathrm{gdc}(b, \phi(p_i^{e_i}))$. The binomial congruence $x^t \equiv b \pmod s$ has solutions if and only if, for each i, $1 \leqslant i \leqslant n$, the index of b modulo $p_i^{e_i}$ is a multiple of d_i. If this condition is satisfied, the congruence has $\prod_{i=1}^n d_i$ solutions.*

Proof.

The theorem follows from the combination of two well-known theorems:

— The congruence $x^t \equiv b \pmod{p_i^{e_i}}$ has solutions if and only if the index of b is a multiple of d_i, in which case it has exactly d_i solutions (see Vinogradov[85, p.81])).

— The number of solutions of $x^t \equiv b \pmod s$ is the product of the numbers of solutions to the separate congruences $x^t \equiv b \pmod{p_i^{e_i}}$, $1 \leqslant i \leqslant n$ (see Vinogradov[85, p.48])).

■

Theorem 8. (see Monier[74]). *If s is composite, then the size of the set $L_{MR}(s)$ of all a, $1 \leqslant a < s$, such that s is a strong a-psp is given by*

$$|L_{MR}(s)| = (1 + (2^{\nu n} - 1)/(2^n - 1)) \prod_{u=1}^n \gcd(s', p_i).$$

Proof.

First let us consider the congruence $a^{s'} \equiv 1 \pmod s$. By theorem 7, the number of solutions is given by

$$\prod_{i=1}^n \gcd(s', \phi(p_i^{e_i})) = \prod_{i=1}^n \gcd(s', p_i - 1) = \prod_{i=1}^n \gcd(s', p_i).$$

The other congruences $a^{s' 2^k} \equiv -1 \pmod s$, $0 \leqslant k < \nu_0$, have solutions if and only if $\gcd(s'2^k, \phi(p_i^{e_i}))$ divides $\mathrm{ind}(-1) = \phi(p_i^{e_i})/2 = p_i^{e_i}(p_i - 1)/2$, for any $1 \leqslant i \leqslant n$, i.e. $k < \nu_i = \nu_2(p_i - 1)$ for $i = 1, \ldots, n$, or equivalently

$$k < \nu = \min_{1 \leqslant i \leqslant n} \{\nu_2(p_i - 1)\}.$$

The number of solutions to this congruence is then

$$\prod_{i=1}^{n} \gcd(s' 2^k, \phi(p_i^{e_i})) = \prod_{i=1}^{n} \gcd(s' 2_k, p_i^{e_i}(p_i - 1)) = 1^{kn} \prod_{i=1}^{n} \gcd(s', p_i - 1).$$

Thus we have

$$|L_{MR}(s)| = (1 + \sum_{k=0}^{v-1} 2^{kn}) \prod_{i=1}^{n} \gcd(s', p_i')$$

which after simplification proves the theorem. ■

We now derive an upper bound on $|L_{MR}(s)|/(s-1)$, which we denote by α_s. For s prime, we have $\alpha_s = 1$. For s composite, we distinguish three cases: s is a prime power, a Carmichael number or neither of them.

1) $s = p^e, e \geqslant 2$. Thus $n = 1$ and then

$$\alpha_s = \frac{1}{1 + p + \ldots + p^{e-1}} \frac{\gcd(s', p')}{p'} \leqslant \frac{1}{1+p}.$$

From $p \geqslant 3$, it follows that $\alpha_s \leqslant 1/4$; equality holds only for $s = 9$.

2) s is a Carmichael number. Then, by theorem 3, s is a product of n distinct primes, at least three. Furthermore, $p_i' | s - 1$. The formula becomes

$$\alpha_s = \left(1 + \frac{2^{vn} - 1}{2^n - 1}\right) \prod_{i=1}^{n} \frac{p_i'}{s-1}.$$

From

$$2^{vn} \prod_{i=1}^{n} p_i' \leqslant \prod_{i=1}^{n} (p_i - 1) \leqslant s - 1,$$

we have

$$\alpha_s \leqslant \left(1 + \frac{2^{vn} - 1}{2^n - 1} \frac{1}{2^{vn}}\right) = 1 - \left(1 - \frac{1}{2^{vn}}\right)\left(1 - \frac{1}{2^n - 1}\right),$$

which is an upper bound decreasing with v and n. Since $n \geqslant 3$ and $v \geqslant 1$, one finds that $\alpha_s \leqslant \frac{1}{4}$. In fact, the limit $\frac{1}{4}$ for α_s is approached by Carmichael's numbers with exactly three prime factors, each being $\equiv 3 \pmod 4$. These numbers satisfy $\alpha_s = \phi(s)/4(s-1)$ and $p_i - 1 = 2p_i'$, $i = 1, 2, 3$. Examples given by Monier[74]) are $s = 8\,911$, with $\alpha_s = 0.2$ and $s = 1\,024\,651$ with $\alpha_s = 0.23478$. Notice that if a prime factor of s is $\not\equiv 3 \pmod 4$ then $\alpha_s \leqslant \frac{1}{8}$. An example is $s = 561$ with $\alpha_s = 70/561$.

3) s is not a prime power and not a Carmichael number. Then there exists a p_i such that $p_i | s$ and $\phi(p_i^{e_i}) \chi s - 1$. Therefore, three cases are to be considered:

— $v_i > v_0$: simple computations allow us to conclude that $\alpha_s \leqslant \phi(s)/4(s-1)$ and thus $\alpha_s < \frac{1}{4}$.

— $p_i' \chi s'$: the conclusion is then that $\alpha_s \leqslant \phi(s)/6(s-1)$ and certainly $\alpha_s < \frac{1}{6}$.

— $e_i > 1$: the bound is identical to the second case.

The bound $\frac{1}{4}$ is very likely to be the best possible. Indeed, for the numbers s of the form $s = (2q+1)(4q+1)$, where q is odd and both $(2q+1)$ and $(4q+1)$ are prime, the value α_s is given by $q/(4q+3)$ and thus can be arbitrarily close to $\frac{1}{4}$ if the set of these numbers s is infinite (open question). Monier[73]) gives as examples $s = 15, 91, 703, 1\,891$ and $497\,503$.

From this lengthy discussion, we obtain

Theorem 9. *If s is composite, then*

$$\alpha_s = \frac{|L_{MR}(s)|}{s-1} \leqslant \frac{1}{4}.$$

■

7. Applications

7.1. *Compositeness tests*

For each odd integer s and each base a, $1 \leqslant a < s$, we define the following *predicates*:

1) $C_{psp}(s, a) = \langle s$ is not an a-psp$\rangle$,
2) $C_{MR}(s, a) = \langle s$ has not the property MR for $a\rangle$,
3) $C_{strong}(s, a) = \langle s$ is not a strong a-psp$\rangle$,
4) $C_{Euler}(s, a) = \langle s$ is not an Euler a-psp$\rangle$.

If s is composite, then we denote the set $\{a | 1 \leqslant a < s, C_{test}(s, a)$ holds$\}$ by $W_{test}(s)$. It is the set of *witnesses* of s's compositeness. The following theorem summarizes the properties of these predicates and sets.

Theorem 10. *For each odd integer s and each base a, $1 \leqslant a < s$,*

1) $C_{psp}(s, a) \Rightarrow C_{Euler}(s, a) \Rightarrow C_{MR}(s, a) \Rightarrow C_{strong}(s, a)$.
2) *If any of these tests holds, then s is composite.*
3) *If s is composite, then*

$$|W_{psp}(s)| \geqslant 0$$

(equality holds for Carmichael's numbers),

$$|W_{Euler}(s)| > \frac{s-1}{2},$$

and

$$|W_{MR}(s)| = |W_{strong}(s)| > \frac{3}{4}(s-1).$$

Proof.

This theorem is a simple application of theorems 1, 4, 6 and 9. The bound on $|W_{Euler}(s)|$ is proved by Baratz[71]) and Monier[74]). ■

The predicate $C_{Euler}(s, a)$ is proposed by Solovay and Strassen[70]). Rabin uses $C_{MR}(s, a)$. The equivalent predicate $C_{strong}(s, a)$ is more convenient since it does not require the computation of a gcd.

7.2. *Probabilistic primality tests*

A *probabilistic algorithm* is based on a predicate $C(s, a)$ defined for each odd integer s and each base a, $1 \leqslant a < s$. This predicate $C(s, a)$ will have the following properties:

1) The number s is composite if and only if $C(s, a)$ holds for some a.
2) The running time of a program which computes $C(s, a)$ is short, i.e. bounded by a polynomial in $\ln s$.
3) When s is composite, let $L(s)$ be the set of integers a for which $C(s, a)$ does not hold. Then $|L(s)|/(s-1) \leqslant \frac{1}{2}$, i.e. at least half the a's must be witnesses of s's compositeness.

Such probabilistic algorithms have the following general form. To test whether s is prime, produce a sequence of k independent random integers $\{a_i\}$, with $1 \leqslant a_i < s$. For each a_i, check whether the predicate $C(s, a_i)$ holds. If so, then s is composite, otherwise it is not certain that s is prime but we declare s to be prime with a probability at least equal to $1 - 2^{-k}$.

The probabilistic primality tests of Solovay-Strassen[70]) and Miller-Rabin[67,68]) have this form with the use of the predicates $C_{Euler}(s, a)$ and $C_{MR}(s, a)$ respectively. Rabin proposes a value between 10 and 30 for k.

7.3. *Use of strong a-psp notion for small calculators*

An easy primality test suitable for use on a small calculator consists in just verifying the (strong) pseudoprimality and then using a table of composite (strong) pseudoprimes.

Lehmer[79]), Norman[86]) and Poulet[80]) have constructed tables containing the composite 2-psp's up to 100 000 000. These tables are very lengthy.

On the other hand, Pomerance, Selfridge and Wagstaff[82]) find, after performing exhaustive computations, that the smallest composite a-psp is

$$\begin{aligned} N_1 &= 2\,047 = 23 \cdot 89, && \text{for } a = 2,\\ N_2 &= 1\,373\,653 = 829 \cdot 1\,657, && \text{for } a = 2 \text{ and } 3,\\ N_3 &= 25\,326\,001 = 2\,251 \cdot 11\,251, && \text{for } a = 2, 3 \text{ and } 5,\\ N_6 &= 3\,215\,031\,751 = 151 \cdot 751 \cdot 28\,351, && \text{for } a = 2, 3, 5 \text{ and } 7. \end{aligned}$$

Only N_3, $N_4 = 161\,304\,001 = 7\,333 \cdot 21\,997$ and $N_6 = 960\,946\,321 = 11\,717 \cdot 82\,013$ are composite strong 2, 3, 5-psp and $< 10^9$. The number N_6 is the only composite strong a-psp for $a = 2, 3, 5, 7$ which does not exceed $2.5 \cdot 10^{10}$.

Using this last idea Monier[8, appendix]) describes a program for the pocket calculator HP-41C: each number $< 10^9$ is tested for primality in less than two minutes.

8. Primality tests

We present here the results which are of interest for generating large numbers for cryptographic applications, based on the theorems of Brillhart, Lehmer and Selfridge[22]).

Let us recall that a number s, which is a candidate for primality, is of the form $s = kF + 1$, where F is a large even number, k is very small in comparison with s and F. The primality tests we have selected allow us to establish constraints on F so as to improve the efficiency of the algorithms for generating prime numbers.

The contents of this section are as follows. The first part contains theorems in which $s - 1$ is assumed to e completely factored. The second part contains theorems which use only partial factorizations of $s - 1$. All these theorems are based on the following converse of Fermat's theorem.

Theorem 11. (see Carmichael[87, p.56])). *If there exists an integer a such that $a^{s-1} \equiv 1 \pmod s$ and if further there does not exist an integer v less than $s - 1$ such that $a^v \equiv 1 \pmod s$, then the integer s is a prime number.* ■

8.1. *Theorems requiring a complete factorization of $s - 1$*

The main theorem Brillhart used for primality testing is due to Lehmer[79]).

Theorem 12. *If there exists an a such that $a^{s-1} \equiv 1 \pmod s$ but $a^{(s-1)/p} \not\equiv 1 \pmod s$ for every prime divisor p of $s - 1$, then s is prime.* ■

However it is difficult to find a small base a for which all the hypotheses of theorem 12 are satisfied. Selfridge observed that these hypotheses can be released to allow a change of base, if needed, for each prime factor of $s - 1$.

Theorem 13. *Let $s - 1 = \prod p_i^{\alpha_i}$, where the p_i's are primes. If for each p_i there exists an a_i such that s is an a_i-psp but $a_i^{(s-1)/p_i} \not\equiv 1 \pmod s$, then s is prime.*

Proof.

Let e_i be the order of $a_i \pmod s$. The hypotheses imply $e_i | s - 1$ but $e_i \nmid (s-1)/p_i$. Hence $p_i^{\alpha_i} | e_i$. But, for each i, $e_i | \phi(s)$, so that $p_i^{\alpha_i} | \phi(s)$, that is $(s - 1) | \phi(s)$. Hence s is prime. ■

To illustrate theorem 13 we note that the primality of

$$s = (2^{104} + 1)/257 = 78\,919\,881\,726\,271\,091\,143\,763\,623\,681$$

where $s - 1 = 2^8 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13 \cdot 17 \cdot 97 \cdot 193 \cdot 241 \cdot 673 \cdot 65\,537 \cdot 22\,253\,377$ can be decided with $a = 3$ for $p = 3, 5, 13, 17, 97, 193, 241, 673, 65\,537$ and $22\,253\,377$, and with $a = 7$ for $p = 7$ and with $a = 11$ for $p = 2$ (see Brillhart and Selfridge[88]).

For each i, the average number of a_i's satisfying $a_i^{s-1} \equiv 1 \pmod{s}$ and $a_i^{(s-1)/p_i} \not\equiv 1 \pmod{s}$ is $(1 - 1/p_i)(s-1)$ if s is prime. In the next theorem, the condition $a^{(s-1)/2} \equiv 1 \pmod{s}$ is used for showing that s is an a-psp.

Theorem 14. *Let $s - 1 = \prod p_i^{\alpha_i}$, where the p_i's are primes. If for each p_i there exists an a_i such that $a_i^{(s-1)/2} \equiv -1 \pmod{s}$ but, for $p_i > 2$, $a_i^{(s-1)/2p_i} \not\equiv -1 \pmod{s}$, then s is prime.*

Proof.

Let us assume $a_i^{(s-1)/2} \equiv -1 \pmod{s}$ and let $a_i^{(s-1)/2p_i} \equiv b_i \not\equiv -1 \pmod{s}$, for each $p_i > 2$. Then $a_i^{(s-1)/p_i} \equiv b_i^2 \not\equiv 1 \pmod{s}$. Indeed we have $-1 \equiv a_i^{(s-1)/2} \equiv b_i^{p_i} \pmod{s}$, which if $b_i^2 \equiv 1 \pmod{s}$ would imply, since p_i is odd, $-1 \equiv b_i^{p_i} \equiv b_i$ (mod s), in contradiction with the second hypothesis. Hence, by theorem 13, s is prime. ■

This theorem is an improvement over theorem 13 in that less calculation is required to complete the primality test. However, for each i, the number of a_i's satisfying $a_i^{(s-1)/2} \equiv -1 \pmod{s}$ and $a_i^{(s-1)/2p_i} \not\equiv -1 \pmod{s}$ is only $(\frac12 - \frac12 p_i)(s-1)$ if s is prime.

Let us remark the uncomplicated nature of these two theorems. A single program can be written to carry out the primality testing without requiring much memory space. Such a program, however, requires more running time than that based on the partial factorization of $s - 1$.

8.2. *Theorems only requiring a partial factorization of $s-1$*

In the special case where a prime factor p of $s-1$ exceeds $s^{\frac12}/2$, the next theorem provides a primality test involving less computation than theorem 14. Only one successful choice of a base a for which the hypothesis relative to p holds is necessary to conclude that s is prime. The other prime factors of $s-1$ can be ignored.

Theorem 15. *Let $s - 1 = mp$, where p is an odd prime such that $2p + 1 > p^{\frac12}$. If there exists an a for which $a^{(s-1)/2} \equiv -1 \pmod{s}$ but $a^{(s-1)/2p} \equiv -1 \pmod{s}$, then s is prime.*

Proof.

Let e be the order of a modulo s. From $a^{s-1} \equiv 1 \pmod{s}$ and $a^{(s-1)/p} \not\equiv 1 \pmod{s}$, we conclude $p \mid e$, whence $p \mid \phi(s)$, since $e \mid \phi(s)$. But $\phi(s) \mid s\prod(q_i - 1)$, where the q_i's are the different prime factors of s and $s = mp+1$. So $p \mid \prod(q_i - 1)$, that is, $p \mid (q_i - 1)$ for some i, say $p \mid (q_1 - 1)$. Thus $q_1 \equiv 1 \pmod{2p}$, and since $s \equiv 1 \pmod{2p}$ too, we have $s/q_1 \equiv 1 \pmod{2p}$. On the other hand, since $q_1 \equiv 1 \pmod{2p}$, we have $q_1 \geqslant 2p + 1 > s^{\frac12}$, from which it follows that $1 \leqslant s/q_1 < s^{\frac12} > 2p + 1$. Therefore, since $s/q_1 \equiv 1 \pmod{2p}$, the only possibility for s/q_1 is 1, and so s is prime. ■

Let us note that the number of a's for which the hypotheses hold is $(\frac12 - \frac12 p)(s-1)$, if s is prime.

Throughout the rest of this section the notation $s - 1 = F_1 R_1$ will be used, where F_1 is the even factored portion of s and R_1 is relatively prime with F_1.

Let us mention the following version of Pocklington's theorem which will be helpful for proving theorem 17.

Theorem 16. *If, for each prime p_i dividing F_1, there exists an a_i such that s is an a_i-psp and $\gcd(a_i^{(s-1)/p_i} - 1, s) = 1$, then each prime divisor of s is $\equiv 1$ (mod F_1).* ■

We present now another theorem, which is superior to theorem 13 in that it requires only that the factored portion F_1 exceeds $(s/2)^{\frac13}$.

Theorem 17. *Assume that, for each prime p_i dividing F_1, there exists an a_i such that s is an a_i-psp and*

$$\gcd(a_i^{(s-1)/p_i} - 1, s) = 1. \tag{8}$$

Let there be given a positive integer m such that $\lambda F_1 + 1 \chi s$ for $1 \leqslant \lambda < m$. If

$$s < (mF_1 + 1)\,[2F_1^2 + (r - m)F_1 + 1], \tag{9}$$

where q and r are defined by $R_1 = 2F_1 q + r$, $1 \leqslant r < 2F_1$, then s is prime if and only if $q = 0$ or $r^2 - 8q$ is not a perfect square (note that $r \neq 0$ since R_1 is odd).

Proof.

The theorem will be proven in the following form: s is composite if and only if $q \neq 0$ and $r^2 - 8q$ is a perfect square. First we prove the necessary condition. From the previous theorem all factors of s are $\equiv 1 \pmod{F_1}$. Thus, since s is composite, we may write $s = (cF_1 + 1)(dF_1 + 1)$ where $c, d \geqslant m$ from the hypotheses, from which we obtain $s - 1 = cdF_1^2 + (c+d)F_1$, and thus $R_1 = cdF_1 + (c + d)$. As F_1 is even and R_1 is odd, this implies that $c + d$ is odd and so cd is even. From $cdF_1 + (c+d) = R_1 = 2F_1 q + r$, we deduce $c + d \equiv r \pmod{2F_1}$ and $c + d - r \geqslant 0$, since $1 \leqslant r < 2F_1$. On the other hand, $(c - m)(d - m) \geqslant 0$ implies $cd \geqslant m(c+d) - m^2$, so that

$$\begin{aligned}(mF_1 + 1)\,[2F_1^2 + (r-m)F_1 + 1] > s &= (cF_1 + 1)(dF_1 + 1) = cdF_1^2 + (c+d)F_1 + 1\\ &\geqslant [m(c+d) - m^2]F_1 + (c+d)F_1 + 1\\ &= (mF_1 + 1)[((c+d) - m)F_1 + 1].\end{aligned}$$

Hence, $2F_1^2 + (r - m)F_1 + 1 > [(c+d) - m]F_1 + 1$, that is, $c + d - r < 2F_1$. It follows that $c + d = r$ and $cd = 2q$. Thus $q \neq 0$ and $r^2 - 8q = (c-d)^2$.

We now prove the sufficient condition. Let $r^2 - 8q = t^2$. We simply calculate

$$\begin{aligned} s &= F_1 R_1 + 1\\ &= F_1(2F_1 q + r) + 1\\ &= [(r^2 - t^2)F_1^2]/4 + rF_1 + 1\\ &= [F_1(r - t)/2 + 1]\,[F_1(r + t)/2 + 1].\end{aligned}$$

Since $q \neq 0$, these two factors are > 1. Hence s is composite. ∎

We now discuss the advantages theorem 17 has for primality testing.

a) The number of prime factors p_i of $s-1$ which we must take into account in the hypotheses of theorem 17 is reduced to a minimum. From (9), $s-1$ needs to be factored at most until $F_1 \geqslant (s/2)^{1/3}$. A further reduction is possible if m is chosen to be > 1 and large enough for (9) to be satisfied. Let us indeed consider the right-hand side of (9) as a function $f(m)$ of the variable m. In the interval defined by $1 \leqslant m \leqslant F_1 + r/2$, the function $f(m)$ is increasing. Thus in any case F_1 must be sufficiently large so that $s < f(F_1 + r/2) = (F_1^2 + rF_1/2 + 1)^2$. The cost of this reduction is the time needed to calculate the trial division of s by $\lambda F_1 + 1$ for $m-1$ values of λ.

b) Let us now discuss the hypotheses (8). If s is prime, they are identical to those given in theorem 13. Hence, if s is prime, for each i the average number of a_i's satisfying (8) is $(1 - 1/p_i)(s-1)$. Thus the larger the prime factors of F_1, the more efficient the primality test based on this theorem. However additional computation is required relative to these hypotheses. But this can be reduced to only one greatest common divisor computation: first, for each i, find an a_i such that $a_i^{(s-1)/p_i} - 1 \equiv b_i \neq 0 \pmod{s}$; then calculate the product $\prod b_i \equiv c \pmod{s}$; and finally, if $c \neq 0$, compute $d = \gcd(c, s)$. If $c = 0$, then some b_i has a prime factor in common with s and so s is composite; if $d \neq 1$, then s is composite too.

c) As the authors have checked on several intervals, the last condition of theorem 17, namely $q = 0$ or $r^2 - 8q$ not a perfect square, does not seem to be very severe. If we denote by $T(R_1, R_2)$ the set $\{s = 2pR_1 + 1 \mid p =$ one of the ten largest primes less than 1 000 000, R an odd number such that $\gcd(p, R) = 1$ and $R_1 \leqslant R \leqslant R_2\}$ then, by exhaustive computations, the following results appear: for the sets T_1 (999 999 900 001, 999 999 999 999) and T_2 (9 999 999 900 001, 9 999 999 999 999), the condition just mentioned is always satisfied, $(|T_1| + |T_2| = 899\,991)$ and for the set T_3 (99 999 999 900 001, 99 999 999 999 999), this condition is violated 9 times $(|T_3| = 450\,000)$. Many other computations give similar results.

From the analysis of the primality criteria above, it appears that some tests are more efficient than other when applied to the problem of generating large prime numbers. Let us recall that the integer s tested for primality is defined to be of the form $s = kF + 1$, where F is a random large even number. If the complete factorization of F is known, the primality test based on theorem 17 is the most efficient. F_1 is then taken to be the minimum even portion of F with sufficiently large prime factors so as to satisfy the hypotheses of theorem 17.

The efficiency of this primality test can still be improved, by requiring F to be of the form $F = 2^n p^\alpha R$, where p is a large prime, p^α is about $(s/2)^{1/3}$ and R is a random odd integer. Then it is sufficient to take $F_1 = 2^n p^\alpha$. If by chance a prime factor of F exceeds $(s^{1/3} - 1)/2$, then theorem 15 should be taken into account to test the primality of s.

Note added in proof (november 1982): Additional references are [105–113].

Acknowledgements

Thanks are due to J. C. Lagarias, A. Odlyzko, H. Williams, Ph. Delsarte and J.-M. Goethals for helpful comments.

Philips Research Laboratory *Brussels, August 1982*

REFERENCES

1) R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, C. ACM 21, pp. 120-126, 1978.
2) H. C. Williams, A modification of the RSA public-key encryption procedure, IEEE Trans. Inform. Theory, IT-26, pp. 726-729, 1980.
3) H. C. Williams and B. Schmid, Some remarks concerning the M.I.T. public-key cryptosystem, BIT 19, pp. 525-538, 1979.
4) C. Couvreur and J.-M. Goethals, The RSA public-key cryptosystem, Philips Research Laboratory, Brussels, Report R427, March 1980.
5) R. K. Guy, How to factor a number, Congruences Numerantium XVI, Proc. fifth Manitoba Conf. on Numerical Math., Winnipeg, pp. 49-89, 1976.
6) D. E. Knuth, The art of computer programming, vol. 2: seminumerical algorithms, Addison-Wesley, Reading, Mass., second edition, 1981.
7) D. E. Knuth and L. Trabb Pardo, Analysis of a simple factorization algorithm, Theoretical Computer Science 3, pp. 321-348, 1976.
8) L. Monier, Algorithmes de factorisation d'entiers, Thesis, Paris-Sud Univ. (Orsay-France), 1980.
9) M. C. Wunderlich, A running time analysis of Brillhart's continued fraction factoring method, in Number theory (Ed. Nathanson), Carbondale, Springer-Verlag, N° 751, 1979.
10) G. H. Hardy and E. M. Wright, An introduction to the theory of numbers, Oxford University Press, London, 4th edition, 1960.
11) J. Hartmanis and H. Shank, On the recognition of primes by automata, J. ACM 15, pp. 382-389, 1968.
12) H. Mann and D. Shanks, A necessary and sufficient condition for primality and its source, J. Com. Th. (A) 13, pp. 131-134, 1972.
13) H. Harborth, Prime number criteria in Pascal's triangle, J. London Math. Soc. (2) 16, pp. 184-190, 1977.
14) M. Davis, Y. Matijasevic and J. Robinson, Hilbert's tenth problem. Diophantine equations: positive aspects of a negative solution, Proc. of Symposia in Pure Mathematics 28, pp. 323-378, 1976.
15) V. Pratt, Every prime has a succinct certificate, SIAM J. Comput. 4, pp. 214-220, 1975.
16) W. H. Mills, A prime-representing function, Bull. Amer. Math. Soc. 53, p. 604, 1947.
17) B. de la Rosa, Primes, powers and partitions, Fibonacci Quart. 16, pp. 518-522, 1978.
18) H. C. Williams, Primality testing on a computer, Ars. Combinations 5, pp. 127-185, 1978.
19) J. P. Buhler, R. E. Crandall and M. A. Penk, Primes of the form $n! \pm 1$ and $2\cdot3\cdot5\cdot\ldots\cdot p \pm 1$, Math. Comp. 38, pp. 639-643, 1982.
20) R. E. Crandall and M. A. Penk, A search for large twin prime pairs, Math. Comp. 33, pp. 383-388, 1979.
21) D. A. Plaisted, Fast verification, testing and generation of large primes, Theoretical Computer Science 9, pp. 1-16, 1979 (errata: id., 14, p. 345).

[22]) J. Brillhart, D. H. Lehmer and J. L. Selfridge, New primality criteria and factorizations of $2^m \pm 1$, Math. Comp. 29, pp. 620-647, 1975.
[23]) R. D. Carmichael, On composite numbers P which satisfy the Fermat congruence $a^{P-1} \equiv 1 \pmod{P}$, Amer. Math. Monthly 19, pp. 22-27, 1912.
[24]) H. C. Williams and J. S. Judd, Some algorithms for prime testing using generalized Lehmer functions, Math. Comp. 30, pp. 867-886, 1976.
[25]) H. C. Williams and R. Holte, Some observations on primality testing, Math. Comp. 32, pp. 905-917, 1978.
[26]) H. C. Williams and J. S. Judd, Determination of the primality of N by using factors of $N^2 \pm 1$, Math. Comp. 30, pp. 157-172, 1976.
[27]) J. Barkley Rosser and L. Schoenfeld, Approximate formulas for some functions of prime numbers, Illinois J. Math. 6, pp. 64-94, 1962.
[28]) J. Bohman, On the number of primes less than a given limit, BIT 12, pp. 576-588, 1972.
[29]) J. Bohman, Some computational results regarding the prime numbers below 2000000000, BIT 13, pp. 242-244, 1973.
) D. C. Mapes, Fast method for computing the number of primes less than a given limit, Math. Comp. 17, pp. 179-185, 1963.
[31]) D. N. Lehmer, List of prime numbers from 1 to 10006721, Publication N° 165, Carnegie Institution of Washington, D.C., 1914; reprinted by Hafner Publishing co., New York, 1956 (errata: Math. Comp. 20, p. 642, 1966).
[32]) M. F. Jones, M. Lal and W. J. Blundon, Statistics on certain large primes, Math. Comp. 21, pp. 103-107, 1967.
[33]) D. Shanks, Quadratic residues and the distribution of primes, Math. Comp. 13, pp. 272-284, 1959.
[34]) G. H. Hardy and J. E. Littlewood, Some problems of "partitio numerorum"; III: on the expression of a number as a sum of primes, Acta Math. 44, pp. 1-70, 1923.
[35]) T. M. Apostol, Introduction to analytic number theory, Springer Verlag, New York, 1976.
[36]) C. Bays and R. H. Hudson, The segmented sieve of Eratosthenes and primes in arithmetic progressions to 10^{12}, BIT 17, pp. 121-127, 1977.
[37]) C. Bays and R. H. Hudson, On the fluctuations of Littlewood for primes of the form $4n \pm 1$, Math. Comp. 32, pp. 281-286, 1978.
[38]) C. Bays and R. H. Hudson, Details of the first region of integers x with $\pi_{3,2}(x) < \pi_{3,1}(x)$, Math. Comp. 32, pp. 571-576, 1978.
[39]) W. W. Rouse Ball and M. S. M. Coxeter, Mathematical recreations and essays, Univ. of Toronto Press, 12th edition, 1974.
[40]) D. Zagier, The first 50 million prime numbers, The Mathematical Intelligencer 1, pp. 7-19, 1977.
[41]) J. H. Cadwell, Large intervals between consecutive primes, Math. Comp. 25, pp. 909-913, 1971.
) S. Weintraub, Distribution of primes between 10^{14} and $10^{14} + 10^8$, Math. Comp. 26, p. 596 (UMT 27), 1972.
[43]) S. Weintraub, Four tables concerning the distribution of primes, Math. Comp. 27, pp. 676-677 (UMT 38), 1973.
[44]) S. Weintraub, A large prime gap, Math. Comp 36, p. 279, 1981.
[45]) D. H. Lehmer, A history of the sieve process, in A history of computing in the twentieth century, A collection of essays, edited by N. Metropolis and al., Academic Press, New York, 1980.
[46]) J. M. Deshouillers, Progrès récents des petits cribles arithmétiques, Séminaire Bourbaki, Springer-Verlag 710, pp. 248-262, 1978.
[47]) M. Halberstam and H. F. Richert, Sieve methods, Academic Press, London, 1974.
[48]) C. Hooley, Application of sieve methods to the theory of numbers, Cambridge Univ. Press, 1976.
[49]) B. A. Chartres, Algorithms 310-311: prime numbers generators 1 and 2, C. ACM 10, pp. 569-570, 1967.
[50]) R. C. Singleton, Algorithms 356-357: prime number generation using treesort principle, C. ACM 12, pp. 563-564, 1969.
[51]) T. C. Wood, Algorithm 35: sieve, C. ACM 4, p. 151, 1961.
[52]) H. G. Mairson, Some new upper bounds on the generation of prime numbers, C. ACM 20, pp. 664-669, 1977.

[53]) D. Gries and J. Misra, A linear sieve algorithm for finding prime numbers, C. ACM 21, pp. 999-1003, 1978.
[54]) P. Pritchard, A sublinear additive sieve for finding prime numbers, C. ACM 24, pp. 18-23, 1980.
[55]) M. K. Wells, Elements of combinatorial computing, Pergamon Press, Oxford, 1971.
[56]) M. C. Wunderlich, Sieving procedures on a digital computer, J. ACM 14, pp. 10-19, 1967.
[57]) S. Weintraub, A compact prime listing, Math. Comp. 28, pp. 855-857, 1974.
[58]) C. L. Baker and F. S. Gruenberger, The first six million prime numbers, The Rand corporation, Santa Monica, published by the Microcard Foundation, Madison, Wisconsin, 1959 (review in Math. Comp. 15, p. 82, 1961).
[59]) D. H. Lehmer, The sieve problem for all-purpose computers, Math. Tables Aids Comput. 7, pp. 6-14, 1953.
[60]) D. H. Lehmer, An announcement concerning the delay line SIEVE DLS-127, Math. Comp. 20, pp. 645-646, 1966.
[61]) D. G. Cantor, G. Estrin, A. S. Fraenkel and R. Turn, A very highspeed digital number sieve, Math. Comp. 16, pp. 141-154, 1962.
[62]) R. P. Brent and E. M. McMillan, Some new algorithms for high-precision computation of Euler's constant, Math. Comp. 34, pp. 305-312, 1980.
[63]) K. L. Appel and J. Barkley Rosser, Table for estimating functions of primes, Communications Research Division, Technical Report N° 4, Institute for Defense Analysis, Princeton, N.J., 1961 (review in Math. Comp. 16, pp. 500-501, 1962).
[64]) L. Schoenfeld, Sharper bounds for the Chebyshev functions $\theta(x)$ and $\psi(x)$. II. Math. Comp. 30, pp. 337-360, 1976.
[65]) S. M. Johnson, An elementary remark on maximal gaps between successive primes, Math. Comp. 19, pp. 675-676, 1965.
[66]) G. L. Miller, Riemann's hypothesis and tests for primality, Proc. 7th annual ACM Symp. Theory of Computing, Albuquerque, N. Mex., pp. 234-239, 1975.
[67]) G. L. Miller, Riemann's hypothesis and tests for primality, J. Comput. Syst. Sci. 13, pp. 300-317, 1976.
[68]) M. O. Rabin, Probabilistic algorithms, in Algorithms and Complexity, J. F. Traub (ed.), Academic Press, New York, pp. 21-40, 1976.
[69]) M. O. Rabin, Probabilistic algorithm for testing primality, Journ. of Number Theory 12, pp. 128-138, 1980.
[70]) R. Solovay and V. Strassen, A fast Monte-Carlo test for primality, SIAM J. Comput. 16, pp. 84-85, 1977 (errata: id., 7, p. 118, 1978).
[71]) A. E. Baratz, An analysis of the Solovay and Strassen test for primality, Report MIT/LCS/TM-108, Lab. for Computer Science, M.I.T., 1978.
[72]) G. J. Chaitin and J. T. Schwartz, A note on Monte-Carlo primality tests and algorithmic information theory, Comm. Pure Appl. Math. 31, pp. 521-527, 1978.
[73]) L. Monier, Evaluation and comparison of two efficient probabilistic primality testing algorithms, Rapport de recherche, N° 20, Lab. Rech. Informat., Paris-Sud Univ. (Orsay-France), June 1978.
[74]) L. Monier, Evaluation and comparison of two efficient probabilistic primality testing algorithms, Theoretical Computer Science 12, pp. 97-108, 1980.
[75]) Cipolla, Annali di Mat. 3, pp. 139-160, 1903.
[76]) J. Chernick, On Fermat's simple theorem, Bull. Am. Math. Soc. 45, pp. 269-274, 1939.
[77]) S. Sispañov, Sobre los numbros pseudo-primos, Boletin Matematico 14, pp. 99-106, 1941.
[78]) O. Ore, Number theory and its history, McGraw-Hill, Inc., New York, 1948.
[79]) D. N. Lehmer, Tests for primality by the converse of Fermat's theorem, Bull. Am. Math. Soc. 33, pp. 327-340, 1927 (errata: Math. Comp. 23, p. 217, 1969).
[80]) P. Poulet, Table des nombres composés vérifiant le théorème de Fermat pour le module 2 jusqu'à 100000000, Sphinx (Brussels) 8, pp. 42-52, 1938 (errata: Math. Comp. 25, pp. 944-945, 1971, MTE 485; 26, p. 814, MTE 497, 1972).
[81]) J. D. Swift, Table of Carmichael numbers to 10^9, Math. Comp. 29, pp. 338-339 (UMT 13), 1975.
[82]) C. Pomerance, J. L. Selfridge and S. S. Wagstaff, The pseudoprimes to $25 \cdot 10^9$, Math. Comp. 35, pp. 1003-1026, 1980.
[83]) C. Pomerance, On the distribution of pseudoprimes, Math. Comp. 37, pp. 587-593, 1981.
[84]) H. C. Pocklington, The determination of the prime or the composite nature of large numbers by Fermat's theorem, Proc. Cambridge Philos. Soc. 18, pp. 29-30, 1914-1916.

85) I. M. Vinogradov, An introduction to the theory of numbers, Pergamon Press, Oxford, 1955 (translation from the russian edition).
86) A. C. Norman, Testing word-sized numbers for primality, SIGSAM Bull. 13, pp. 19-20, 1979.
87) R. D. Carmichael, The theory of numbers, Dover publications, inc., New York, 1914.
88) J. Brillhart and J. L. Selfridge, Some factorizations of $2^n \pm 1$ and related results, Math. Comp. 21, pp. 87-96, 1967 (errata: ib., p. 751).
89) D. H. Lehmer, On the converse of Fermat's theorem, Amer. Math. Monthly 43, pp. 347-354, 1936.
90) D. H. Lehmer, Computer technology applied to the history of numbers, in Studies in number theory, Math. Assoc. Amer. (distr. by Prentice-Hall, Englewood Cliffs, N.J.), pp. 117-151, 1969.
91) M. A. Morrison, A note on primality testing using Lucas sequences, Math. Comp. 29, pp. 181-182, 1975.
92) M. A. Morrison and J. Brillhart, A method of factoring and the factorization of F_7, Math. Comp. 29, pp. 183-205, 1975.
93) L. Adleman and F. T. Leighton, An $O(n^{1/10.89})$ primality testing algorithm, Math. Comp. 36, pp. 261-266, 1981.
94) L. Adleman, On distinguishing prime numbers from composite numbers, Proc. 21st FOCS Conference, Syracuse, NY, pp. 387-406, 1980.
95) C. Pomerance, Recent developments in primality testing, The Mathematical Intelligencer 3, pp. 97-105, 1981.
96) H. W. Lenstra, Jr, Tests de primalité et théorie de Galois, Journées de théorie des nombres, Reims, Mars 1981.
97) H. W. Lenstra, Jr, Primality testing algorithms (after Adleman, Rumely and Williams), Séminaire Bourbaki, 33e année, 1980-1981, n° 576, Marseille, France, Juin 1981.
98) H. Cohen, Tests de primalité d'après Adleman, Rumely, Pomerance et Lenstra, Séminaire de Théorie des nombres, Grenoble, Juin 1981.
99) D. E. G. Malm, On Monte-Carlo primality tests, Notices Amer. Math. Soc. 24, p. A-529, Abstract #77T-A22, 1977.
100) H. W. Lenstra, Jr, Miller's primality test, Inform. Proc. Let. 8, pp. 86-88, 1979.
101) J. Vélu, Tests for primality under the Riemann hypothesis, SIGACT News 10, pp. 58-59, 1978.
102) M. Mignotte, Tests de primalité, Theoretical Computer Science 12, pp. 109-117, 1980.
103) S. Pajunen, On two theorems of Lenstra, Inform. Proc. Let. 11, pp. 224-228, 1980.
104) R. Beillie and S. S. Wagstaff, Lucas pseudoprimes, Math. Comp. 35, pp. 1391-1417, 1980.
105) M. Pollard, An algorithm for testing the primality of any integer, Bull. London Math. Soc. 3, pp. 337-340, 1971.
106) J. M. Pollard, Theorems on factorization and primality testing, Proc. Camb. Phil. Soc. 76, pp. 521-528, 1976.
107) J. L. Selfridge and R. K. Guy, Primality testing with application to small machines, Proc. Washington State Univ., Conf. on Number Theory, Pullman, 1971, pp. 45-51.
108) J. L. Selfridge and M. C. Wunderlich, An efficient algorithm for testing large numbers for primality, Proc. 4th Manitoba Conf. on Numerical Math., Winnipeg, Manitoba, 1974, pp. 109-120.
109) M. C. Wunderlich and J. L. Selfridge, A design for a number theory package with an optimized trial division routine, C. ACM 17, pp. 272-276, 1974.
110) P. Pritchard, Explaining the wheel sieve, Acta Informatice 17, pp. 477-485, 1982.
111) G. Robin, Estimation de la fonction de Tchebychef θ sur le $k^{\text{ème}}$ nombre premier et grandes valeurs de le fonction $\omega(n)$, nombre de diviseurs premiers de N, Acta Arithmetica, to appear, 1983.
112) H. Cohen and H. W. Lenstra, Jr, Report 82-18, Mathematisch Instituut, U. of Amsterdam, October 1982.
113) L. Adleman, C. Pomerance and R. Rumely, On distinguishing prime numbers from composite numbers, Annals of Math., to appear.

Philips J. Res. 37, 265-276, 1982

MODELLING OF METAL VAPOUR NOBLE GAS DISCHARGES IN THE TRANSITION REGION

by M. J. C. VAN GEMERT*), O. P. VAN DRIEL**) and J. MEZGER

Abstract

Modelling of metal vapour noble gas discharges has been performed using a hypothetical gas system consisting of metal atoms with a low ionization potential and noble gas atoms with a much higher ionization potential. Both the metal and the noble gas atoms are assumed to have only a ground state and an ionization level. Numerical calculations of the electric field-discharge current (*E-I*) curves reveal the existence of a low and a high field region. The intermediate region can either be multivalued or single valued in *E*, depending on whether or not the metal vapour ionization rate is much larger than the noble gas ionization rate.

1. Introduction

Electrical discharge characteristics showing multivalued behaviour have long been known in the literature[1-6]). For example a maximum for the discharge current exists for a low-pressure mercury arc[1,2]) while low-pressure discharges in mixtures of Cs–Ar[3]), and of Na–Ne[4]) show characteristics multivalued in the voltage. For the latter type of discharges modelling has so far been limited to metal vapour ionization only[6]). Even in such a restricted model the electric field is calculated to be multivalued; because of metal vapour depletion a maximum in the current is found, since noble gas ionization is not taken into account.

When noble gas ionization is taken into consideration the electric field-discharge current (*E-I*) characteristics consist of a low field region at low currents, where metal vapour ionization predominates, and a high field region at higher currents where noble gas ionization is predominant. A current maximum is then not found.

The purpose of this work is to present an analysis of the *E–I* characteristic in the intermediate region, where both metal vapour ionization and noble gas ionization play a part. To this end a simple discharge model is developed in sec. 2. The numerical results are presented in sec. 3. A discussion in given in

*) Present address: Department of Medical Technology, St. Joseph Hospital, Eindhoven, The Netherlands.

**) Present address: Elcoma Division, Nijmegen, The Netherlands.

Errata to:

An introduction to fast generation of large prime numbers, *Philips J. Res.* 37, 231-264, 1982, by C. Couvreur and J. J. Quisquater.

	read ...	instead of ...
p. 233, col. 6	Williams and al.[25,26]	Williams and al.[23,25]
p. 238, line 3	*suppress it*	due to Hardy and Littlewood[34]
p. 239, col. 6	$\dfrac{\pi(x)}{\phi(a)}$	$\pi(x)$
p. 242, col. 2	282	292
p. 246, line 11	$Q(x) \sim \exp\left(-\sum_{p \in P_s} \frac{1}{p}\right)$,	$Q(x) \sim \exp\left(-\sum_{p} \in p_{:} \frac{1}{p}\right)$,
p. 247, table 10	$5 \cdot 10^5$	5.10^5
p. 248, line 25	$\text{lcm}\,(\phi(p_1^{e_1}), \ldots, \phi(p_n^{e_n}))$,	$1\ \text{cm}\,(\phi(p_n^{e_n}))$,
p. 249, line 6	see Apostol[35,p.114]).	see Apostol[35 , p.114]).
p. 251, line 10	for some k, $0 \leqslant k < \nu_0$.	for some k, $0 < k < \nu_0$.
line 25	$a^{(s-1)/2^{k-1}} \equiv 1 \pmod s$,	$a^{(s-1)/2^k} \equiv 1 \pmod s$,
line 29	$a^{s' 2^{\nu_0 - k}} \equiv -1 \pmod s$	$a^{2^{\nu_0 - k_s'}} \equiv -1 \pmod s$
last line	$0 \leqslant k < \nu_0$, then	$0 \leqslant k \leqslant \nu_0$, then
p. 252, line 1	$a^{(s-1)/2^t} \not\equiv 1$	$a^{(s-1)/2^l} \not\equiv 1$
pp. 252-4-5-7-9	$\not\mid$ (does not divide)	χ (chi)
p. 253, line 10	$d_i = \gcd(b, \phi(p_i^{e_i}))$	$d_i = \text{gdc}(b, \phi(p_i^{e_i}))$
line 23	$\prod_{i=1}^{n} \gcd(s', p_i')$.	$\prod_{u=1}^{n} \gcd(s', p_i')$.
p. 254, line 1	$\gcd(s' 2^k, p_i^{e_i}(p_i - 1)) = 2^{kn}$	$\gcd(s' 2_k, p_i^{e_i}(p_i - 1)) = 1^{kn}$
line 3	$\prod_{i=1}^{n} \gcd(s', p_i')$.	$\prod_{i-1}^{n} \gcd(s', p_i')$.
p. 257, line 14	to be completely	to e completely
p. 258, line 37	$< s^{\frac{1}{2}} < 2p + 1$.	$< s^{\frac{1}{2}} > 2p + 1$.
p. 261, ref. [18])	Ars Combinatoria	Ars. Combinations
p. 263, ref. [77])	Sispanov	Sispañov
p. 264, ref. [104])	R. Baillie	R. Beillie
ref. [110])	Acta Informatica	Acta Informatice

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

263

A. M. Odlyzko (Ed.)

Advances in Cryptology – CRYPTO '86

Proceedings

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo

PUBLIC-KEY SYSTEMS BASED ON THE DIFFICULTY OF TAMPERING
(Is there a difference between DES and RSA?)

Yvo Desmedt * and Jean-Jacques Quisquater **

(*) Katholieke Universiteit Leuven, ESAT, Belgium [1].
Current address: Université de Montréal, Dépt. IRO, Case postale 6128, succursale A, Montréal (Québec), H3C 3J7 Canada.
(**) Philips Research Laboratory Brussels, Avenue Van Becelaere, 2, B-1170 Brussels, Belgium.

Abstract

This paper proposes several public key systems which security is based on the tamperfreeness of a device instead of the computational complexity of a trapdoor one-way function. The first identity-based cryptosystem to protect privacy is presented.

EXTENDED ABSTRACT

1 Introduction

We first give three main motives for this paper and overview the presented ideas.

Since the invention of public-key systems by Diffie and Hellman almost all public-key systems proposed were based on some computational hard problems (e.g. factoring). It was however shown that it is not easy to design a secure public-key system based on computational hard problems. Examples of failures are the Lu-Lee system, the Merkle-Hellman knapsack scheme (and others) and the Matsumoto-Imai scheme. If we remark that the McEliece scheme is not enough analysed to be used, there do not exist fast public-key systems (the speed of RSA is today less than 64 kbit/sec.). This is one of the main reasons to come up with other public-key systems.

Bennett and Brassard remarked that it is not necessary to use computational complexity to design a public-key system. As an example they started from the uncertainty principle, which claims that some physical problems are very hard to solve (impossible to measure). Bennett and Brassard mentioned that their system would remain secure if NP=P and if factoring would be easy. However the cryptosystems they proposed are today impractical. One can conclude that a second reason for this paper is to design cryptosystems which are not based on the assumption that trapdoor one-way functions exist.

The authenticity of the public key is a major problem in the set-up of a secure cryptosystem, certainly in the case of a large network. A nice solution was proposed by Shamir in 1984 called "*identity-based cryptosystem*". Instead of using the public key of the receiver (to encrypt in order to protect the privacy of a message), the name of the receiver is used as public key. The secret key of each user was calculated by an authority at the start-up of the system. (It is not excluded that the authority destroys itself after the start-up of the system.) Public-key systems, identity-based cryptosystems and their key generation are systematically explained in Fig. 1.

---

[1] This research was done while the author was aangesteld navorser NFWO at the Katholieke Universiteit Leuven

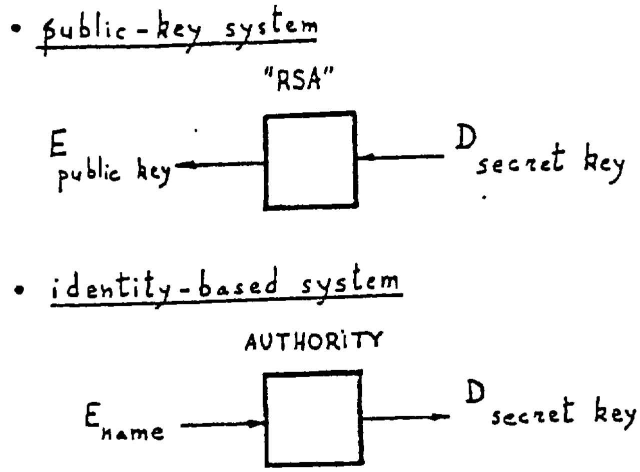


Figure 1: Key generation for public-key and identity-based systems

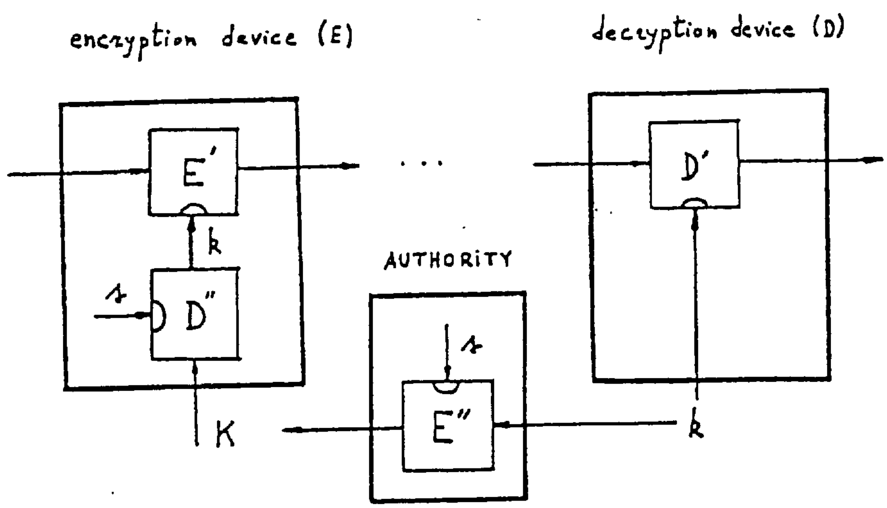


Figure 2: A first implementation of a public-key system

In our paper we st
that it is possible to m
on the complexity of
sideration that it is n
conventional ones. Wi
would become unsecur
nent can simply steal
from this second assur
NPL becomes comple
that each identificatio
card (see also Section

Given two convent
tions we propose in ou
cryptosystem to prote

2 Public keys

2.1 The basic id

Let us give an exampl
encryption and decryp
Special cases use the al
for D' and D''. To o
device (corresponding
operation D) and a sy
(corresponding to the
obtains his correspond
is nothing but E'' wit
device G is tamperfree
key s is used in all de

2.2 Two implem

We now discuss two in
and Fig. 3).

In the first examp
In fact here D is equa
key K. The system
box E first D'' is use
calculation is done ins
The outside world. In
The encryption of me

The described sch
authenticity of messa
public key of the recei
that nevertheless the
To sign the sender us

In our paper we start from the assumptions that hard conventional systems exist and that it is possible to make tamperfree devices. Remark that the first assumption is based on the complexity of algorithms, but seems acceptable, certainly if one takes into consideration that it is much harder to build trapdoor one-way public-key systems than conventional ones. Without the second assumption a lot of modern uses of cryptography would become unsecure. Indeed a secure system must be tamperfree otherwise an opponent can simply steal the secret key used in the system. Several practical systems start from this second assumption. *E.g.*, a software copyright protection system proposed by NPL becomes completely insecure if tamperfree devices can not be build. Remark too that each identification method is at least partially based on some tamperfree system or card (see also Section 5).

Given two conventional cryptosystems and the existence of tamperfree implementations we propose in our full paper several public-key systems, and the first identity-based cryptosystem to protect privacy.

2 Public keys

2.1 The basic idea

Let us give an example of such a system. From now on we call E', D', E'' and D'' the encryption and decryption of respectively the first and second conventional cryptosystems. Special cases use the algorithm DES in encryption mode for E' and E'' or decryption mode for D' and D''. To obtain a public-key system three devices are used: an encryption device (corresponding to the operation E), a decryption device (corresponding to the operation D) and a system which generates the public key starting from the secret key (corresponding to the operation G). Each user of the system generates a secret key k. He obtains his corresponding public key K by applying G on k, or $K = G(k)$. The device G is nothing but E'' with a supersecret key s (which in the best case nobody knows). The device G is tamperfree so that it is hard to find the key s. In this example the supersecret key s is used in all devices G.

2.2 Two implementations of such a public-key system

We now discuss two implementations to obtain such a public-key system (see also Fig. 2 and Fig. 3).

In the first example (see also Fig. 2) the decryption device (D) uses the secret key. In fact here D is equal to D'. The encryption device (E) uses as a black box the public key K. The system E is build up using E' and D''. The box E is tamperfree. In the box E first D'' is used to find k, or $k = D''(K)$ using the supersecret key s. This last calculation is done inside E, and no trace of this calculation and its result can leak out to the outside world. In other words because the device E is tamperfree it is hard to find k. The encryption of messages is done by E' using the key k.

The described scheme can be used to protect, as a public-key system, the privacy and authenticity of messages as well to sign. To protect privacy the sender uses E with the public key of the receiver (although the receiver uses D with his secret key). Remark again that nevertheless the sender uses in fact the secret key of the sender, he cannot access it. To sign the sender uses D with his secret key (evidently redundancy is introduced in the

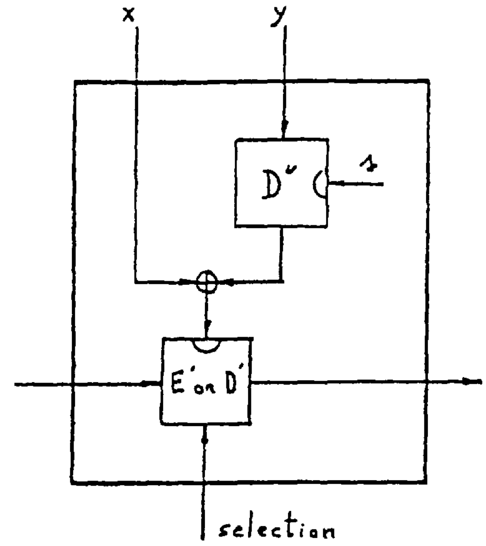


Figure 3: A second implementation of a public-key system

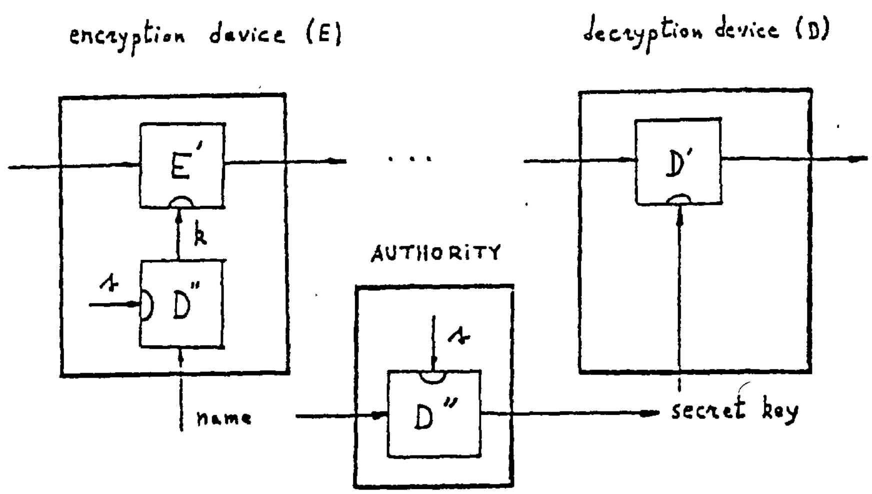


Figure 4: The first identity-based system to protect privacy

message). The receive
sender is the only one

The second impler
same tamperfree devic
system in some words
device used in the sys
and a corresponding p
The device (T) contai
send an encrypted mes
and applies his secret
the input y. To decry
and applies his secret
input y. In these two
two parties. There are
a.s.o. Let us remark th
system) together with
encryption and the dec
public-key system).

3 Identity-ba

By modifying a little bi
the public key of some
machine G now is modi
the input of G is the n
output is the secret key
G are controlled by an
give as input somethin
father, name of compan
the authentication of t
open by Adi Shamir, to

4 Security

In this section only nec
discussed. Sufficient cor

The system E'' has
s by cryptanalytic meth
adaptive chosen text att
could be set up, certain
identity-based cryptosy
several users (which hav

Evidently the crypto

Another necessary c
term introduced by Da
a weak key there is no

message). The receiver can check the signature (using the mentioned redundancy). The sender is the only one who could generate that signature.

The second implementation has the advantage that each user in the system has the same tamperfree device for encryption as well as for decryption. Let us describe such a system in some words. For this paragraph, we refer to Fig. 3. Let (T) be the tamperfree device used in the system. As for the first system, each user i generates a secret key k_i and a corresponding public key K_i. For that he uses the device G as already discussed. The device (T) contains E', D', D'' and the supersecret key s as described in Fig. 3. To send an encrypted message to a user B, a user A uses the device (T) in mode encryption and applies his secret key k_A to the input x and the public key K_B of the user B to the input y. To decrypt this message the user B uses the device (T) in mode decryption and applies his secret key k_B to the input x and the public key K_A of the user A to the input y. In these two phases, the effective key in use is the same but is unknown to the two parties. There are many variants to this scheme with the possibility of a session key, a.s.o. Let us remark that using a symmetric cryptosystem (sometimes called conventional system) together with such a symmetric implementation (the devices are the same for the encryption and the decryption) leads to an asymmetric cryptosystem (sometimes called public-key system).

3 Identity–based cryptosystem

By modifying a little bit previous examples it is no longer necessary to use public keys (or the public key of somebody is equal to his name or identification). The key generation machine G now is modified. The system G now uses D'' (with the supersecret key s) and the input of G is the name (or a sufficient identification of the person to be unique), the output is the secret key of the user (see also Fig. 4). In order to avoid frauds the uses of G are controlled by an authority. Each user can use G only once, and is only allowed to give as input something that corresponds with his identification (birth day, name of his father, name of company, ...). This is a first advantage because it avoids in large networks the authentication of the public key. This technique gives a first solution to a problem open by Adi Shamir, to propose an identity-based cryptosystem to protect privacy.

4 Security

In this section only necessary conditions in order to obtain a secure implementation are discussed. Sufficient conditions are still under research.

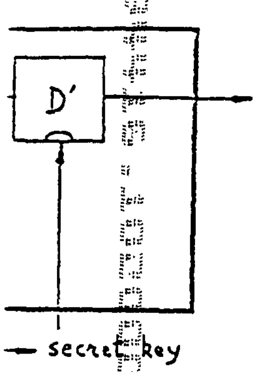
The system E'' has to be a secure cryptosystem such that all attacks fail in finding s by cryptanalytic methods. Therefore it is necessary that E'' is secure e.g. against an adaptive chosen text attack. The reader could wonder how an adaptive chosen text attack could be set up, certainly if an authority limits the use of the device G (as in the case of identity-based cryptosystem). The answer is that the adaptive aspect can be obtained if several users (which have e.g. special names) collaborate.

Evidently the cryptosystems E', D' and D'' have also to be secure cryptosystems.

Another necessary condition is that the system may not have (or use) weak keys (a term introduced by Davies related to weak keys in DES) or similar weaknesses. Using a weak key there is no difference between an encryption and a decryption operation.

-key system

yption device (D)



tect privacy

Indeed an asymmetry is required to obtain public-key systems. If not, this implies that everybody can generate signatures of an opponent using his public key, because E' will in fact internally use the secret key of the opponent and for weak keys this E' operation is the same as the D' operation. In general in order to protect signatures (with the described scheme) it must be hard to generate outputs of D' starting from outputs of E'. So semi-weak keys are also dangerous. The same remark holds for the protection of privacy. Otherwise everybody could decrypt message send to Bob, using Bob's public key for a similar reason.

5 Advantages, disadvantages and other aspects

A major advantage of the discussed systems is the speed. Using DES (and dropping weak keys) much faster public-key systems can be made. An important disadvantage of the system is that everybody who knows s can attack all users! However in some cases such a property is desired (by the authority), as in the case of communications between persons of a same company (*e.g.* a bank). In this context we remark that the key distribution problem in some large companies (when a normal conventional system is used), can be hard to solve.

Remark also that in previous discussions one can *e.g.* replace the supersecret key s, by some secret function. In the discussed example E', D', E'' and D'' are public known conventional algorithms. It is trivial to understand that the same holds if E', D', E'' and D'' are secret. In other words if some organization promotes secret algorithms, key distribution centers can be avoided and one can use the described public-key method. Indeed in order to maintain the secrecy of the used secret algorithms, the devices must be at least tamperfree.

Finally one can question that the described system is really a public-key system. To solve this problem one can use the well known Turing test. Suppose DES and RSA are used (to be mathematically correct n DESes are used with n different keys), is it then possible to find in polynomial time (as function of n) if DES or RSA is used? It is well known that the answer is yes, using the Jacobi symbol in a known plaintext attack. In a secure implementation of RSA and DES it must be hard to make a difference between real random and the ciphertext in polynomial time. As a consequence if DES (in such public-key system) and RSA are used in a secure implementation, no difference can be observed in polynomial time.

Remark that in a part of our paper on the importance of good key scheduling schemes (1985, CRYPTO '85), we did not obtain a real public-key system as we do here, moreover, some of our assumptions there are the opposite of some assumptions here.

It is not too hard to find better schemes which satisfy some desired properties, some of these other schemes are still under research. For instance, in the context of tamperfree devices, it is possible to design claw-free functions with conventional cryptoalgorithms and thus to have very fast algorithms to sign documents (Rivest, Goldwasser, Micali, Goldreich).

Another advantage is that the above idea of identity-based cryptosystem can be used in a protocol in order to protect passports. Let us again start from the assumption that tamperfree devices and that conventional cryptosystems exist, where the decryption operation can not be obtained by applying polynomially the encryption operation. *Remark that the assumption of tamperfree devices is also necessary in Shamir's protocol* (presented

at the same conference). I
secret (the square roots ir
very busy businessman o
to clone themselves, in o
aspects, for which the or
tween the identity of the
to know the secret corres
Our identification pro
type of algorithm is used a
we use the identity-based
distributes to other count
visitor (e.g. Alice) tells t
country which she visits
Israel and the name (ide
Belgium generates then s
key (obtained from her c
gives to Belgium. If both
this system is that 200 di
The advantage is that eac
made by other countries.
research.

6 Open Proble

A main open problem is
and which security is not
Another open proble
in Section 5. Does there
security is based on tan
different supersecret s for
if the computational pro
reverse situation happen
The authors have th
related.

Remarks

Other works, more or le
S. M. Matyas and C. H.

Acknowledgement

The authors are grateful

at the same conference). Indeed if an owner of a passport is able to find his corresponding secret (the square roots in Shamir's protocol), there is no protection against cloning. For very busy businessman or consultants or researchers it can be an important advantage to clone themselves, in order that the cloned one handles the public relation and other aspects, for which the original persons are too busy. If a difference has to be made between the identity of the person and his cloned version, the person himself is not allowed to know the secret corresponding with his secret. So tamperfree devices are necessary.

Our identification protocol is very similar to the one of Shamir, except that a different type of algorithm is used and that the country that is visited generates the random. Again we use the identity-based cryptosystem to protect signatures. Each country (*e.g.* Israel) distributes to other countries the E devices, containing their supersecret s. During use, a visitor (*e.g.* Alice) tells the officials her nationality (e.g. Israelian) and her identity. The country which she visits (e.g. Belgium) then uses the tamperfree device obtained from Israel and the name (identity) of Alice is used as key by that country (*e.g.* Belgium). Belgium generates then some random t and gives $E(t)$ to Alice. If Alice knows her secret key (obtained from her country: Israel), she is able to decrypt it and obtain t, which she gives to Belgium. If both t's match Belgium accepts Alice identity. The disadvantage of this system is that 200 different kinds of machines are necessary (each for each country). The advantage is that each country relies on their own technology to avoid false passports made by other countries. A proof for the security of the discussed protocol is still under research.

6 Open Problems

A main open problem is to find an identity-based cryptosystem which protects privacy and which security is not based on the assumption of the existence of tamperfree devices.

Another open problem is to overcome the problem of the supersecret key s, mentioned in Section 5. Does there exist an identity-based cryptosystem to protect privacy which security is based on tamperfree devices and computational complexity and which use different supersecret s for different users. In other words that system would remain secure if the computational problem is solved, but the tamperfreeness is still valid, or if the reverse situation happened.

The authors have the impression that both mentioned open problems are strongly related.

Remarks

Other works, more or less related to this one, were made by M. E. Smid, R. E. Lennon, S. M. Matyas and C. H. Meyer, H. Beker and M. Walker.

Acknowledgement

The authors are grateful to Adi Shamir for the discussions related to Section 6.

not shown in the Figure, the dominant-mode wavelength of the short laser also shifted one mode spacing towards longer wavelengths relative to the spontaneous emission peak under DC operation just below threshold, probably owing to an increase in junction temperature.[4] Note that the build-up time of the dominant mode is significantly faster in the short-cavity and the type-B ridge-waveguide lasers than in the type-A device.

The time-dependent output of these lasers, at discrete wavelengths and in real time, is shown in Fig. 2, where for clarity the evolution of four individual shots are shown (clean traces) in comparison with several thousand pulses (smeared traces).

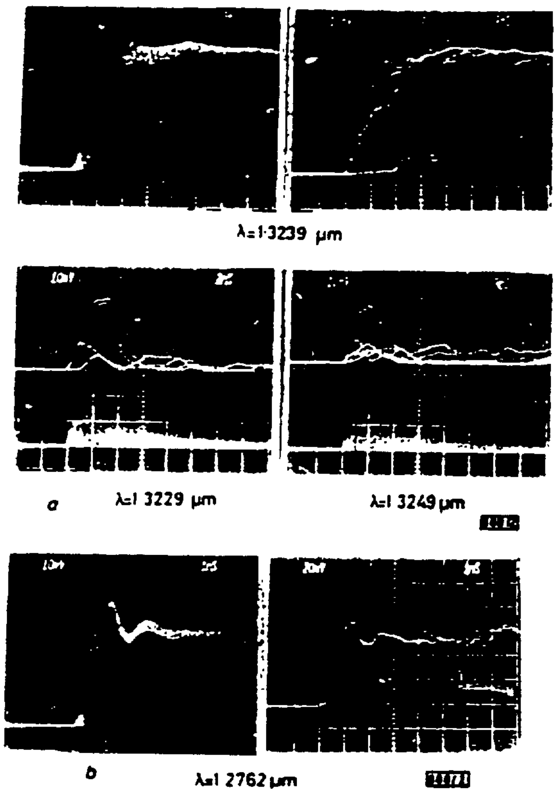


Fig. 2 *Transient response of (a) type-A standard-length ridge-waveguide laser and (b) short-cavity laser*

Both traces of a few thousand scans and four individual scans are shown. The wavelength indicates the spectrometer setting at which the traces were recorded. Type-B ridge-waveguide laser behaved similarly to the short-cavity laser

Note that, for the type-A ridge-waveguide laser, Fig. 2*a*, the individual pulse of any one mode can start at different times and go through different evolution paths.[5,6] The secondary modes decay while the dominant mode increases to its steady-state value in about 6 ns. A similar display of the output of the dominant mode of the type-B or the short-cavity laser is reproduced in Fig. 2*b*, which shows far less pulse-to-pulse variation.

We conclude that any laser with genuinely stable single-mode output, whether achieved by design or by accident (as, for example, by a buried periodic ripple providing wavelength-selective feedback), leads to transient behaviour compatible with modulation at high bit rates.

The origin of the intensity fluctuations is the spontaneous emission.[3,7-9] Even when biased slightly below threshold, the number of spontaneous photons at each longitudinal mode wavelength is significant,[3] and the instantaneous spectrum just before application of the current step has large fluctuations. Subsequent to the arrival of the current step, each mode builds up at different rates from these fluctuations until the stimulated emission of the dominant mode finally takes over. It has been pointed out that, if the side mode initially contains significant power, it takes several nanoseconds for it to decay. In a short-cavity laser, however, the decay of the side modes is faster, allowing faster build-up of the dominant mode.

We note again that the jitter in the equipment was less than 50 ps. Thus the displayed random fluctuations (partition of the optical energy among longitudinal modes as a function of time) represent a direct observation of the mode partition noise in real time.

We are indebted to J. A. Copeland, E. A. J. Marcatili and S. E. Miller for unpublished information, and to N. K. Cheung and A. Tomita for the use of equipment.

PAO-LO LIU

Bell Laboratories
Holmdel, New Jersey 07733, USA

T. P. LEE
C. A. BURRUS
I. P. KAMINOW
J.-S. KO

Bell Laboratories
Crawford Hill Laboratory
Holmdel, New Jersey 07733, USA

31st August 1982

References

1 KAMINOW, I. P., NAHORY, R. E., STULTZ, L. W., and DEWINTER, J. C.: 'Performance of an improved InGaAsP ridge waveguide laser at 1·3 μm', *Electron. Lett.*, 1981, 17, pp. 318–320
2 BURRUS, C. A., LEE, T. P., and DENTAI, A. G.: 'Short-cavity single-mode 1·3 μm InGaAsP lasers with evaporated high-reflectivity mirrors', *ibid.*, 1981, 17, pp. 954–956
3 LEE, T. P., BURRUS, C. A., COPELAND, T. A., DENTAI, A. G., and MARCUSE, D.: 'Short-cavity InGaAsP injection lasers: Dependence of mode spectra and single-longitudinal power on cavity length', *IEEE J. Quantum Electron.*, 1982, QE-18, pp. 1101–1113
4 NAGANO, M., and KASAHARA, K.: 'Dynamic properties of transverse junction stripe lasers', *ibid.*, 1977, QE-13, pp. 632–637
5 MACHIDA, S., TSUCHIYA, H., and ITO, T.: 'Single-mode optical fiber cable transmission experiment at 0·85 μm wavelength', *Rev. Electr. Commun. Lab.*, 1979, 27, pp. 599–610
6 HENNING, I.: 'Technique for measuring true time-resolved spectra of a semiconductor laser', *Electron. Lett.*, 1982, 18, pp. 368–369
7 MCCUMBER, D. E.: 'Intensity fluctuations in the output of cw laser oscillators I', *Phys. Rev.*, 1966, 141, pp. 306–322
8 JÄCKEL, H., and GUEKOS, G.: 'High frequency intensity noise spectra of axial groups in the radiation from cw GaAlAs diode lasers', *Opt. & Quantum Electron.*, 1977, 9, pp. 233–239
9 ITO, T., MACHIDA, S., NAWATA, K., and IKEGAMI . 'Intensity fluctuations in each longitudinal mode of a multimode AlGaAs laser', *IEEE J. Quantum Electron.* 1977, QE-13, pp. 574–579

0013-5194/82/210904-02$1.50/0

FAST DECIPHERMENT ALGORITHM FOR RSA PUBLIC-KEY CRYPTOSYSTEM

Indexing terms: Codes, Cryptography, Public-key cryptosystem, RSA

A fast algorithm is presented for deciphering cryptograms involved in the public-key cryptosystem proposed by Rivest, Shamir and Adleman. The deciphering method is based on the Chinese remainder theorem and on improved modular multiplication algorithms.

Introduction: Among the published public-key cryptosystems, the scheme proposed by Rivest, Shamir and Adleman[1] (usually referred to as the RSA or MIT cryptosystem) seems to be the most attractive for many applications. Its security is based on the fact that any known successful cryptanalytic attack has the same complexity as the factorisation of a large composite number:[2,3] at this time, no very efficient method of factoring is known. However, a frequently quoted disadvantage of the RSA cryptosystem is the relative time complexity

of its operations (discrete exponentia... modulo a large integer) as compared to conventional systems such as the DES.[4,5]

In this letter a fast algorithm is presented for deciphering cryptograms in the RSA system, which is about 4–8 times faster than the classical algorithm for computing a modular exponentiation.[2] This algorithm is based on the Chinese remainder theorem and on improved modular multiplications.

RSA scheme: Let an RSA box be a small electronic device[3] the memory of which contains two large prime numbers p and q. These numbers have been generated by the RSA box itself and are accessible to nobody. The product $r = pq$ has been computed and a random integer e which is relatively prime with both $p - 1$ and $q - 1$ has been generated too. The RSA box has also precomputed the only integer $d < r$ such that

$$ed \equiv 1 (\text{mod } (p-1)(q-1))$$

The enciphering key consists of the pair (e, r), possibly listed in a directory. The deciphering key is the pair (d, r) and is kept secret in the RSA box.

If a user wants to send a private message M to the owner of this RSA box, he proceeds as follows:

(i) He retrieves the (public) enciphering key (e, r).

(ii) He breaks the message M into a sequence of blocks $(m_1, \ldots, m_i, \ldots, m_k)$, where each block is represented as an integer m_i between 0 and $r - 1$.

(iii) He transmits the cryptograms $(c_1, \ldots, c_i, \ldots, c_k)$, where $c_i = E(m_i) = m_i^e \; (\text{mod } r)$.

The RSA box can decipher the cryptograms c_i by computing $D(c_i) = c_i^d (\text{mod } r) = m_i$. Hence the message M is recovered by the owner of the RSA box when the whole sequence $(c_1, \ldots, c_k)$ is deciphered.

Fast deciphering algorithm: Classically, as the quantities m, r, e and d would be about 500 or 600 bits long,[4,6,7] the enciphering and the deciphering processes require up to several hundred multiplications of integers of this length. The enciphering key can be as short as 2 bits,[2,4] but, for avoiding attacks by enumerative techniques, the deciphering key requires the maximum length. However, the deciphering process can be expedited. Before describing the fast deciphering algorithm, some notations[2,8] must be introduced.

Let us consider the following residues of the quantities m, c and d:

$$c_1 = c(\text{mod } p) \qquad c_2 = c(\text{mod } q)$$
$$d_1 = d(\text{mod } p-1) \qquad d_2 = d(\text{mod } q-1)$$
$$m_1 = m(\text{mod } p) = c_1^{d_1}(\text{mod } p)$$
$$m_2 = m(\text{mod } q) = c_2^{d_2}(\text{mod } q)$$

since the message m and the cryptogram c are related by $m = c^d (\text{mod } r)$.

Given p and q, $p < q$, let A be a constant integer such that $0 < A < q - 1$ and $A_p \equiv 1 (\text{mod } q)$. This constant is obtained by applying Euclid's algorithm[2] for computing gcd (p, q). By using the Chinese remainder theorem it is easily observed that m satisfies

$$m = [((m_2 + q - m_1)A)(\text{mod } q)]p + m_1 \qquad (1)$$

Hence, to decipher the cryptogram c, the algorithm first computes $m_1 = c_1^{d_1}(\text{mod } p)$ and $m_2 = c_2^{d_2}(\text{mod } q)$ rather than computing $m = c^d (\text{mod } r)$ classically. The quantities p, q, c_1, c_2, d_1 and d_2 are now only about 300 bits long. This permits one to reduce the time complexity to about a quarter. Moreover the two computations may be done in parallel. To recover the message m, it remains to compute expr. 1.

Let us remark that the exponents d_1 and d_2 may be chosen to be greater than $p - 1$ and $q - 1$; that does not affect the result. But if the (binary) weight of the exponent is smaller, then the modular exponentiation becomes possibly faster.

Even so, the most time-consuming part of the deciphering scheme remains the modular exponentiations. A modular exponentiation algorithm for computing $P = c^d (\text{mod } p)$ is described in the Appendix. This algorithm is distinguished from the classical ones. Many simplifications are made due to the context in which it is implemented. For example, the modular multiplications by c are reduced to a sequence of table look-ups and accumulations.[9] Also the number P is mostly required to be at most $n = \lceil \log_2 p \rceil$ bits long[10] and not necessarily smaller than p. This explains that only the most significant bit of P, and not the integer P itself, is tested before a possible reduction of P. So the reductions modulo p are made as few as possible. These reductions are also very simplified by the precomputations of the integers Q and R. Finally, let us remark that this algorithm does away with the integer division.

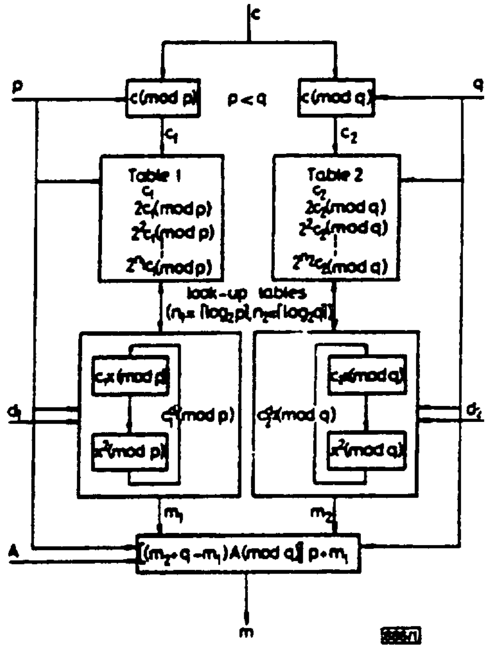


Fig. 1

Fig. 1 is a functional diagram of the deciphering process of the RSA cryptosystem, using this improved modular exponentiation algorithm and computing $m_1 = c_1^{d_1}(\text{mod } p)$ and $m_2 = c_2^{d_2}(\text{mod } q)$ in parallel.

If the lengths of p and q are about 256 bits, then Tables 1 and 2 use $2 \times (256)^2$ bits $\simeq$ 128 kbits: this value is within the range of current technology. Faster implementation is still possible with additional memory of the expressions $(2^{i-1} + 2^i)c(\text{mod } p)$ in both Tables.

We would like to mention that Krishnamurthy and Ramachandran[11] have independently proposed to use the Chinese remainder theorem for computing modular exponentiations in their conventional cryptosystems.

Acknowledgments: We would like to thank J.-M. Goethals for helpful comments.

Appendix: Let p be an integer, > 1, with exactly n bits, i.e. $n = \lceil \log_2 p \rceil$. An n-bit number d is represented as $[d_{n-1} \ldots d_1 d_0]$. The following algorithm computes the modular exponentiation.

Procedure MODULAR EXPONENTIATION (c, d, p): given the integers c, d and p, where $0 \le c < p$, $0 \le d < p - 1$, the procedure computes the integer $P = c^d (\text{mod } p)$, $0 \le P < p$.

Initialisation: $Q \leftarrow 2^n - p$; $P \leftarrow 1$;

Step 1: **for** $i = n - 1, n - 2, \ldots, 1$

1.1 **if** $P_{n-1} = 1$ **then** $P \leftarrow$ REDUCTION(P)
1.2 $P \leftarrow$ MODMUL(P, P, p, P)
1.3 **if** $d_i = 1$ **then** $P \leftarrow$ MODMULCO(P, p, table, P);

Step 2: if $P_{n-1} = 1$ **then** $P \leftarrow$ REDUCTION(P);

Return P

The procedures used in the above algorithm can be described as follows.

Procedure REDUCTION (P): given the n-bit integer P, $0 \leq P < 2^n$, and the precomputed (global variable) integer $Q = 2^n - p$, this procedure returns the value $P(\text{mod } p)$, between 0 and $p - 1$.

Initialisation: $R \leftarrow P + Q$;

if $R_n = 1$ **then** $P \leftarrow [R_{n-1} \ldots R_0]$;

Return P

Procedure MODMUL (x, y, p, P): given the integers x, y and p, $0 \leq x < p$, $0 \leq y < 2^n$, this procedure computes the integer $P = x \cdot y(\text{mod } p)$. The integer P is a $(n+1)$-bit number $[P_n P_{n-1} \ldots P_1 P_0]$ but as output, P verifies $0 \leq P < 2^n$.

Initialisation: $R \leftarrow Q + x$; $P \leftarrow 0$;

for $i = n - 1, n - 2, \ldots, 1$

1. $P \leftarrow$ one left shift of P
2. **if** $y_i = 1$ **then**
   if $P_n = 1$ **then** $P \leftarrow [P_{n-1} \ldots P_0] + R$ **else** $P \leftarrow P + x$
3. **if** $P_n = 1$ **then** $P \leftarrow [P_{n-1} \ldots P_0] + Q$
4. **if** $P_n = 1$ **then** $P \leftarrow [P_{n-1} \ldots P_0] + Q$;

Return P

Procedure LOOK-UP TABLE (c, n, p, table): given the integers c, n and p, $0 \leq c < p$, this procedure computes the sequence $c, 2c, 2^2c, \ldots, 2^{n-1}c$, each value being stored modulo p in a table. This table is used by MODMULCO.

Initialisation: $P \leftarrow c$; table(0) $\leftarrow c$;

for $i = 1, 2, \ \ , n - 1$

1. $P \leftarrow$ one left shift of P
2. **if** $P_n = 1$ **then** $P \leftarrow [P_{n-1} \ldots P_0] + Q$
3. **if** $P_{n-1} = 1$ **then** $P \leftarrow$ REDUCTION(P)
4. table(i) $\leftarrow P$;

Return

Procedure MODMULCO (x, p, table, P): given x, $0 \leq x < 2^n$, p and the table generated by LOOK-UP TABLE for the integer c, this procedure returns the value $P = c \cdot x(\text{mod } p)$, $0 \leq P < 2^n$.

Initialisation: $P \leftarrow 0$;

for $i = 0, 1, 2, \ldots, n - 1$

if $x_i = 1$ **then**

1. $P \leftarrow P +$ table(i)
2. **if** $P_n = 1$ **then** $P \leftarrow [P_{n-1} \ldots P_0] + Q$;

Return P

J.-J. QUISQUATER
C. COUVREUR
Philips Research Laboratory
Av. Van Becelaere 2, Box 8
B-1170 Brussels, Belgium

27th August 1982

References

1 RIVEST, R. L., SHAMIR, A., and ADLEMAN, L.: 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, 1978, 21, pp. 120–126
2 KNUTH, D. E.: 'The art of computer programming. Vol. 2: seminumerical algorithms' (Addison-Wesley, Reading, Mass., 2nd edn., 1981)
3 WILLIAMS, H. C.: 'A modification of the RSA public-key encryption procedure', *IEEE Trans.*, 1980, IT-26, pp. 726–729
4 MICHELMAN, E. H.: 'The design and operation of public-key cryptosystems', *NCC*, 1979, pp. 305–311
5 SCHANNING, B. P.: 'Data encryption with public-key distribution'. Proc. Eascon 1979, IEEE, pp. 653–660
6 DAVIES, D. W., PRICE, W. L., and PARKIN, G. I.: 'An evaluation of public-key cryptosystems'. NPL report, CTU1 (revised), April 1980
7 RIVEST, R. L.: 'A description of a single-chip implementation of the RSA cipher', *Lambda*, 4th quarter 1980, pp. 14–18
8 SZABÓ, N. S., and TANAKA, R. I.: 'Residue arithmetic and its applications to computer technology' (McGraw-Hill, New York, 1967)
9 HENRY, P. S.: 'Fast decryption algorithm for the knapsack cryptographic system', *Bell Syst. Tech. J.*, 1981, 60, pp. 767–773
10 BLAKLEY, G. R.: 'A computer algorithm for calculating the product *AB* modulo *M*'. Research report, Department of Mathematics, Texas A & M University, Texas, USA
11 KRISHNAMURTHY, E. V., and RAMACHANDRAN, V.: 'A cryptographic system based on finite field transforms', *Proc. Ind. Acad. Sci. (Math. Sci.)*, 1980, 89, pp. 75–93
12 AHO, A. V., HOPCROFT, J. E., and ULLMAN, J. D.: 'The design and analysis of computer algorithms' (Addison-Wesley, Reading, Mass., 1974)
13 WILLONER, R., and I-NGO CHEN: 'An algorithm for modular exponentiation'. Proc. 5th symp. on computer arithmetic, IEEE Computer society, 1981, pp. 135–138

0013-5194/82/210905-03\$1.50/0

MEASUREMENT OF POLARISATION MODE DISPERSION IN ELLIPTICAL-CORE SINGLE-MODE FIBRES AT 1·3 μm

Indexing terms: Optical fibres, Polarisation, Dispersion

Polarisation mode delay differences in three single-mode fibres were measured interferometrically at 1·3 μm with a resolution below 25 fs. Polarisation mode dispersion increases strongly with core ellipticity.

Introduction: Polarisation mode dispersion may be a limiting factor in high-capacity single-mode optical-fibre transmission systems[1,2] that will be operated most likely at about 1·3 μm wavelength, where the material dispersion is minimum. We report here on polarisation mode dispersion measurements carried out interferometrically at 1·3 μm. This is to complement some related recently published results[3,4] that concentrated on the shorter wavelengths around 0·85 μm. The results illustrate the strong dependence of the polarisation mode dispersion on the fibre core ellipticity. Some special features of our measurement method and set-up resulted in an improved resolution of below ± 25 fs delay time difference.

Measurement set-up: For our measurements we used the interferometric method described by Mochizuki *et al.* (see Fig. 1 in their paper[3]). A temperature stabilised quaternary semiconductor laser, model HLD 5400 (Hitachi), emitting at 1·300 μm was used as optical source. Its spectral profile is shown in Fig. 1.

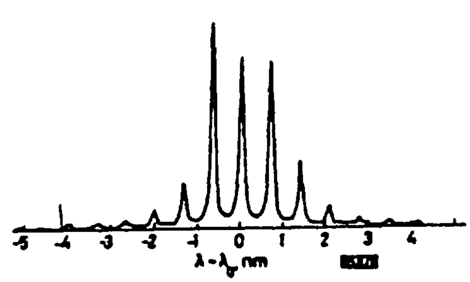


Fig. 1 *Spectral profile of laser diode used (HLD 5400)*
Operating conditions: 28·8 mA, 298 K; centre wavelength: λ_0 = 1300·0 nm

not shown in the Figure, the dominant-mode wavelength of the short laser also shifted one mode spacing towards longer wavelengths relative to the spontaneous emission peak under DC operation just below threshold, probably owing to an increase in junction temperature.[4] Note that the build-up time of the dominant mode is significantly faster in the short-cavity and the type-B ridge-waveguide lasers than in the type-A device.

The time-dependent output of these lasers, at discrete wavelengths and in real time, is shown in Fig. 2, where for clarity the evolution of four individual shots are shown (clean traces) in comparison with several thousand pulses (smeared traces).

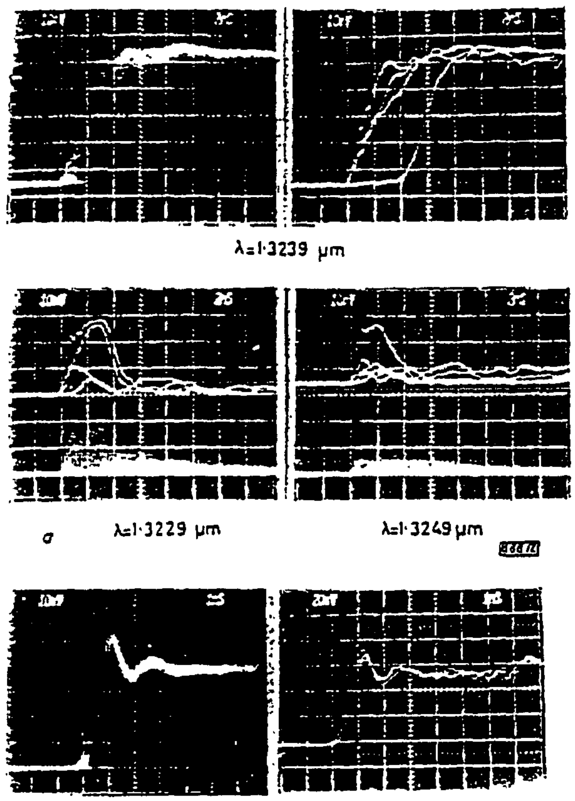


Fig. 2 *Transient response of (a) type-A standard-length ridge-waveguide laser and (b) short-cavity laser*

Both traces of a few thousand scans and four individual scans are shown. The wavelength indicates the spectrometer setting at which the traces were recorded. Type-B ridge-waveguide laser behaved similarly to the short-cavity laser

Note that, for the type-A ridge-waveguide laser, Fig. 2*a*, the individual pulse of any one mode can start at different times and go through different evolution paths.[5,6] The secondary modes decay while the dominant mode increases to its steady-state value in about 6 ns. A similar display of the output of the dominant mode of the type-B or the short-cavity laser is reproduced in Fig. 2*b*, which shows far less pulse-to-pulse variation.

We conclude that any laser with genuinely stable single-mode output, whether achieved by design or by accident (as, for example, by a buried periodic ripple providing wavelength-selective feedback), leads to transient behaviour compatible with modulation at high bit rates.

The origin of the intensity fluctuations is the spontaneous emission.[3,7–9] Even when biased slightly below threshold, the number of spontaneous photons at each longitudinal mode wavelength is significant,[3] and the instantaneous spectrum just before application of the current step has large fluctuations. Subsequent to the arrival of the current step, each mode builds up at different rates from these fluctuations until the stimulated emission of the dominant mode finally takes over. It has been pointed out that, if the side mode initially contains significant power, it takes several nanoseconds for it to decay. In a short-cavity laser, however, the decay of the side modes is faster, allowing faster build-up of the dominant mode.

We note again that the jitter in the equipment was less than 50 ps. Thus the displayed random fluctuations (partition of the optical energy among longitudinal modes as a function of time) represent a direct observation of the mode partition noise in real time.

We are indebted to J. A. Copeland, E. A. J. Marcatili and S. E. Miller for unpublished information, and to N. K. Cheung and A. Tomita for the use of equipment.

PAO-LO LIU

31st August 1982

Bell Laboratories
Holmdel, New Jersey 07733, USA

T. P. LEE
C. A. BURRUS
I. P. KAMINOW
J.-S. KO

Bell Laboratories
Crawford Hill Laboratory
Holmdel, New Jersey 07733, USA

References

1 KAMINOW, I. P., NAHORY, R. E., STULTZ, L. W., and DEWINTER, J. C.: 'Performance of an improved InGaAsP ridge waveguide laser at 1·3 µm', *Electron. Lett.*, 1981, **17**, pp. 318–320

2 BURRUS, C. A., LEE, T. P., and DENTAI, A. G.: 'Short-cavity single-mode 1·3 µm InGaAsP lasers with evaporated high-reflectivity mirrors', *ibid.*, 1981, **17**, pp. 954–956

3 LEE, T. P., BURRUS, C. A., COPELAND, T. A., DENTAI, A. G., and MARCUSE, D.: 'Short-cavity InGaAsP injection lasers: Dependence of mode spectra and single-longitudinal power on cavity length', *IEEE J. Quantum Electron.*, 1982, QE-18, pp. 1101–1113

4 NAGANO, M., and KASAHARA, K.: 'Dynamic properties of transverse junction stripe lasers', *ibid.*, 1977, QE-13, pp. 632–637

5 MACHIDA, S., TSUCHIYA, H., and ITO, T.: 'Single-mode optical fiber cable transmission experiment at 0·85 µm wavelength', *Rev. Electr. Commun. Lab.*, 1979, **27**, pp. 599–610

6 HENNING, I.: 'Technique for measuring true time-resolved spectra of a semiconductor laser', *Electron. Lett.*, 1982, **18**, pp. 368–369

7 MCCUMBER, D. E.: 'Intensity fluctuations in the output of cw laser oscillators I', *Phys. Rev.*, 1966, **141**, pp. 306–322

8 JÄCKEL, H., and GUEKOS, G.: 'High frequency intensity noise spectra of axial groups in the radiation from cw GaAlAs diode lasers', *Opt. & Quantum Electron.*, 1977, **9**, pp. 233–239

9 ITO, T., MACHIDA, S., NAWATA, K., and IKEGAMI, T.: 'Intensity fluctuations in each longitudinal mode of a multimode AlGaAs laser', *IEEE J. Quantum Electron.*, 1977, QE-13, pp. 574–579

0013-5194/82/210904-02\$1.50/0

FAST DECIPHERMENT ALGORITHM FOR RSA PUBLIC-KEY CRYPTOSYSTEM

Indexing terms: Codes, Cryptography, Public-key cryptosystem, RSA

A fast algorithm is presented for deciphering cryptograms involved in the public-key cryptosystem proposed by Rivest, Shamir and Adleman. The deciphering method is based on the Chinese remainder theorem and on improved modular multiplication algorithms.

Introduction: Among the published public-key cryptosystems, the scheme proposed by Rivest, Shamir and Adleman[1] (usually referred to as the RSA or MIT cryptosystem) seems to be the most attractive for many applications. Its security is based on the fact that any known successful cryptanalytic attack has the same complexity as the factorisation of a large composite number:[2,3] at this time, no very efficient method of factoring is known. However, a frequently quoted disadvantage of the RSA cryptosystem is the relative time complexity

of its operations (discrete exponentiation modulo a large integer) as compared to conventional systems such as the DES.[4,5]

In this letter a fast algorithm is presented for deciphering cryptograms in the RSA system, which is about 4–8 times faster than the classical algorithm for computing a modular exponentiation.[1] This algorithm is based on the Chinese remainder theorem and on improved modular multiplications.

RSA scheme: Let an RSA box be a small electronic device[2] the memory of which contains two large prime numbers p and q. These numbers have been generated by the RSA box itself and are accessible to nobody. The product $r = pq$ has been computed and a random integer e which is relatively prime with both $p-1$ and $q-1$ has been generated too. The RSA box has also precomputed the only integer $d < r$ such that

$$ed \equiv 1(\mathrm{mod}\ (p-1)(q-1))$$

The enciphering key consists of the pair (e, r), possibly listed in a directory. The deciphering key is the pair (d, r) and is kept secret in the RSA box.

If a user wants to send a private message M to the owner of this RSA box, he proceeds as follows:

(i) He retrieves the (public) enciphering key (e, r).

(ii) He breaks the message M into a sequence of blocks $(m_1, \ldots, m_i, \ldots, m_k)$, where each block is represented as an integer m_i between 0 and $r-1$.

(iii) He transmits the cryptograms $(c_1, \ldots, c_i, \ldots, c_k)$, where $c_i = E(m_i) = m_i^e\ (\mathrm{mod}\ r)$.

The RSA box can decipher the cryptograms c_i by computing $D(c_i) = c_i^d(\mathrm{mod}\ r) = m_i$. Hence the message M is recovered by the owner of the RSA box when the whole sequence $(c_1, \ldots, c_k)$ is deciphered.

Fast deciphering algorithm: Classically, as the quantities m, r, e and d would be about 500 or 600 bits long,[4,6,7] the enciphering and the deciphering processes require up to several hundred multiplications of integers of this length. The enciphering key can be as short as 2 bits,[2,4] but, for avoiding attacks by enumerative techniques, the deciphering key requires the maximum length. However, the deciphering process can be expedited. Before describing the fast deciphering algorithm, some notations[2,8] must be introduced.

Let us consider the following residues of the quantities m, c and d:

$$c_1 = c(\mathrm{mod}\ p) \qquad c_2 = c(\mathrm{mod}\ q)$$
$$d_1 = d(\mathrm{mod}\ p-1) \qquad d_2 = d(\mathrm{mod}\ q-1)$$
$$m_1 = m(\mathrm{mod}\ p) = c_1^{d_1}(\mathrm{mod}\ p)$$
$$m_2 = m(\mathrm{mod}\ q) = c_2^{d_2}(\mathrm{mod}\ q)$$

since the message m and the cryptogram c are related by $m = c^d(\mathrm{mod}\ r)$.

Given p and q, $p < q$, let A be a constant integer such that $0 < A < q-1$ and $A_p \equiv 1(\mathrm{mod}\ q)$. This constant is obtained by applying Euclid's algorithm[2] for computing gcd (p, q). By using the Chinese remainder theorem it is easily observed that m satisfies

$$m = [((m_2 + q - m_1)A)(\mathrm{mod}\ q)]p + m_1 \qquad (1)$$

Hence, to decipher the cryptogram c, the algorithm first computes $m_1 = c_1^{d_1}(\mathrm{mod}\ p)$ and $m_2 = c_2^{d_2}(\mathrm{mod}\ q)$ rather than computing $m = c^d(\mathrm{mod}\ r)$ classically. The quantities p, q, c_1, c_2, d_1 and d_2 are now only about 300 bits long. This permits one to reduce the time complexity to about a quarter. Moreover the two computations may be done in parallel. To recover the message m, it remains to compute expr. 1.

Let us remark that the exponents d_1 and d_2 may be chosen to be greater than $p-1$ and $q-1$; that does not affect the result. But if the (binary) weight of the exponent is smaller,

Even so, the most time-consuming part of the deciphering scheme remains the modular exponentiations. A modular exponentiation algorithm for computing $P = c^d(\mathrm{mod}\ p)$ is described in the Appendix. This algorithm is distinguished from the classical ones. Many simplifications are made due to the context in which it is implemented. For example, the modular multiplications by c are reduced to a sequence of table look-ups and accumulations.[9] Also the number P is mostly required to be at most $n = \lceil \log_2 p \rceil$ bits long[10] and not necessarily smaller than p. This explains that only the most significant bit of P, and not the integer P itself, is tested before a possible reduction of P. So the reductions modulo p are made as few as possible. These reductions are also very simplified by the precomputations of the integers Q and R. Finally, let us remark that this algorithm does away with the integer division.

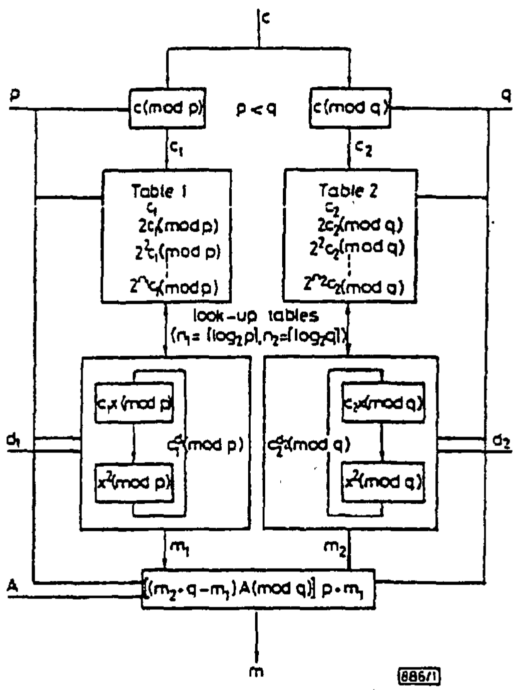


Fig. 1

Fig. 1 is a functional diagram of the deciphering process of the RSA cryptosystem, using this improved modular exponentiation algorithm and computing $m_1 = c_1^{d_1}(\mathrm{mod}\ p)$ and $m_2 = c_2^{d_2}(\mathrm{mod}\ q)$ in parallel.

If the lengths of p and q are about 256 bits, then Tables 1 and 2 use $2 \times (256)^2$ bits $\simeq 128$ kbits: this value is within the range of current technology. Faster implementation is still possible with additional memory of the expressions $(2^{i-1} + 2^i)c(\mathrm{mod}\ p)$ in both Tables.

We would like to mention that Krishnamurthy and Ramachandran[11] have independently proposed to use the Chinese remainder theorem for computing modular exponentiations in their conventional cryptosystems.

Acknowledgments: We would like to thank J.-M. Goethals for helpful comments.

Appendix: Let p be an integer, > 1, with exactly n bits, i.e. $n = \lceil \log_2 p \rceil$. An n-bit number d is represented as $[d_{n-1} \ldots d_1 d_0]$. The following algorithm computes the modular exponentiation.

Procedure MODULAR EXPONENTIATION (c, d, p): given the integers c, d and p, where $0 \le c < p$, $0 \le d < p-1$, the procedure computes the integer $P = c^d(\mathrm{mod}\ p)$, $0 \le P < p$.

Initialisation: $Q \leftarrow 2^n - p$; $P \leftarrow 1$;

Step 1: for $i = n-1, n-2, \ldots, 1$

1.1 if $P_{n-1} = 1$ then $P \leftarrow$ REDUCTION(P)

1.2 $P \leftarrow$ MODMUL(P, P, p, P)

Step 2: if $P_{n-1} = 1$ then $P \leftarrow$ REDUC⁻ ⁻N(P);

Return P

The procedures used in the above algorithm can be described as follows.

Procedure REDUCTION (P): given the n-bit integer P, $0 \le P < 2^n$, and the precomputed (global variable) integer $Q = 2^n - p$, this procedure returns the value $P(\text{mod } p)$, between 0 and $p-1$.

Initialisation: $R \leftarrow P + Q$;

if $R_n = 1$ then $P \leftarrow [R_{n-1} \dots R_0]$;

Return P

Procedure MODMUL (x, y, p, P): given the integers x, y and p, $0 \le x < p$, $0 \le y < 2^n$, this procedure computes the integer $P = x \cdot y(\text{mod } p)$. The integer P is a $(n+1)$-bit number $[P_n P_{n-1} \dots P_1 P_0]$ but as output, P verifies $0 \le P < 2^n$.

Initialisation: $R \leftarrow Q + x$; $P \leftarrow 0$;

for $i = n-1, n-2, \dots, 1$

1. $P \leftarrow$ one left shift of P
2. if $y_i = 1$ then
   if $P_n = 1$ then $P \leftarrow [P_{n-1} \dots P_0] + R$ else $P \leftarrow P + x$
3. if $P_n = 1$ then $P \leftarrow [P_{n-1} \dots P_0] + Q$
4. if $P_n = 1$ then $P \leftarrow [P_{n-1} \dots P_0] + Q$;

Return P

Procedure LOOK-UP TABLE (c, n, p, table): given the integers c, n and p, $0 \le c < p$, this procedure computes the sequence $c, 2c, 2^2c, \dots, 2^{n-1}c$, each value being stored modulo p in a table. This table is used by MODMULCO.

Initialisation: $P \leftarrow c$; table(0) $\leftarrow c$;

for $i = 1, 2, \dots, n-1$

1. $P \leftarrow$ one left shift of P
2. if $P_n = 1$ then $P \leftarrow [P_{n-1} \dots P_0] + Q$
3. if $P_{n-1} = 1$ then $P \leftarrow$ REDUCTION(P)
4. table(i) $\leftarrow P$;

Return

Procedure MODMULCO (x, p, table, P): given x, $0 \le x < 2^n$, p and the table generated by LOOK-UP TABLE for the integer c, this procedure returns the value $P = c \cdot x(\text{mod } p)$, $0 \le P < 2^n$.

Initialisation: $P \leftarrow 0$;

for $i = 0, 1, 2, \dots, n-1$

if $x_i = 1$ then

1. $P \leftarrow P +$ table(i)
2. if $P_n = 1$ then $P \leftarrow [P_{n-1} \dots P_0] + Q$;

Return P

J.-J. QUISQUATER
C. COUVREUR

Philips Research Laboratory
Av. Van Becelaere 2, Box 8
B-1170 Brussels, Belgium

27th August 1982

References

1 RIVEST, R. L., SHAMIR, A., and ADLEMAN, L.: 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, 1978, 21, pp. 120–126
2 KNUTH, D. E.: 'The art of computer programming, Vol. 2: seminumerical algorithms' (Addison-Wesley, Reading, Mass., 2nd edn., 1981)
3 WILLIAMS, H. C.: 'A mc ition of the RSA public-key encryption procedure', *IEEE Trans.*, 1980, IT-26, pp. 726–729
4 MICHELMAN, E. H.: 'The design and operation of public-key cryptosystems', *NCC*, 1979, pp. 305–311
5 SCHANNING, B. P.: 'Data encryption with public-key distribution', Proc. Eascon 1979, IEEE, pp. 653–660
6 DAVIES, D. W., PRICE, W. L., and PARKIN, G. I.: 'An evaluation of public-key cryptosystems'. NPL report, CTU1 (revised), April 1980
7 RIVEST, R. L.: 'A description of a single-chip implementation of the RSA cipher', *Lambda*, 4th quarter 1980, pp. 14–18
8 SZABÓ, N. S., and TANAKA, R. I.: 'Residue arithmetic and its applications to computer technology' (McGraw-Hill, New York, 1967)
9 HENRY, P. S.: 'Fast decryption algorithm for the knapsack cryptographic system', *Bell Syst. Tech. J.*, 1981, 60, pp. 767–773
10 BLAKLEY, G. R.: 'A computer algorithm for calculating the product *AB* modulo *M*'. Research report, Department of Mathematics, Texas A & M University, Texas, USA
11 KRISHNAMURTHY, E. V., and RAMACHANDRAN, V.: 'A cryptographic system based on finite field transforms', *Proc. Ind. Acad. Sci. (Math. Sci.)*, 1980, 89, pp. 75–93
12 AHO, A. V., HOPCROFT, J. E., and ULLMAN, J. D.: 'The design and analysis of computer algorithms' (Addison-Wesley, Reading, Mass., 1974)
13 WILLONER, R., and I-NGO CHEN: 'An algorithm for modular exponentiation'. Proc. 5th symp. on computer arithmetic, IEEE Computer society, 1981, pp. 135–138

0013-5194/82/210905-03$1.50/0

MEASUREMENT OF POLARISATION MODE DISPERSION IN ELLIPTICAL-CORE SINGLE-MODE FIBRES AT 1·3 μm

Indexing terms: Optical fibres, Polarisation, Dispersion

Polarisation mode delay differences in three single-mode fibres were measured interferometrically at 1·3 μm with a resolution below 25 fs. Polarisation mode dispersion increases strongly with core ellipticity.

Introduction: Polarisation mode dispersion may be a limiting factor in high-capacity single-mode optical-fibre transmission systems[1,2] that will be operated most likely at about 1·3 μm wavelength, where the material dispersion is minimum. We report here on polarisation mode dispersion measurements carried out interferometrically at 1·3 μm. This is to complement some related recently published results[3,4] that concentrated on the shorter wavelengths around 0·85 μm. The results illustrate the strong dependence of the polarisation mode dispersion on the fibre core ellipticity. Some special features of our measurement method and set-up resulted in an improved resolution of below ± 25 fs delay time difference.

Measurement set-up: For our measurements we used the interferometric method described by Mochizuki *et al.* (see Fig. 1 in their paper[3]). A temperature stabilised quaternary semiconductor laser, model HLD 5400 (Hitachi), emitting at 1·300 μm was used as optical source. Its spectral profile is shown in Fig. 1.

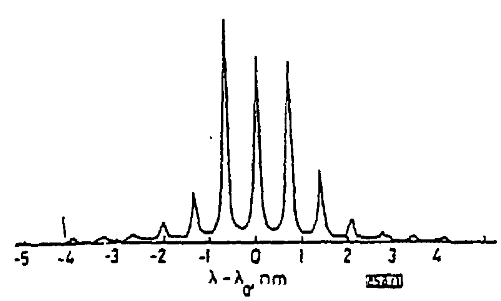


Fig. 1 *Spectral profile of laser diode used (HLD 5400)*
Operating conditions: 28·8 mA, 298 K; centre wavelength: λ_0 = 1300·0 nm

CC Express Mail No. EL636302715

Cetin Kaya

High-Speed RSA Implementation

Çetin Kaya Koç
Koc@ece.orst.edu

RSA Laboratories
RSA Data Security, Inc.
100 Marine Parkway, Suite 500
Redwood City, CA 94065-1031

Version 2.0 – November 1994

Contents

Preface

This report is written for people who are interested in implementing modular exponentiation based cryptosystems. These include the RSA algorithm, the Diffie-Hellman key exchange scheme, the ElGamal algorithm, and the recently proposed Digital Signature Standard (DSS) of the National Institute for Standards and Technology. The emphasis of the report is on the underlying mathematics, algorithms, and their running time analyses. The report does not include any actual code; however, we have selected the algorithms which are particularly suitable for microprocessor and signal processor implementations. It is our aim and hope that the report will close the gap between the mathematics of the modular exponentiation operation and its actual implementation on a general purpose processor.

Chapter 1

The RSA Cryptosystem

1.1 The RSA Algorithm

The RSA algorithm was invented by Rivest, Shamir, and Adleman [41]. Let p and q be two distinct large random primes. The modulus n is the product of these two primes: $n = pq$. Euler's totient function of n is given by

$$\phi(n) = (p-1)(q-1) \ .$$

Now, select a number $1 < e < \phi(n)$ such that

$$\gcd(e, \phi(n)) = 1 \ ,$$

and compute d with

$$d = e^{-1} \bmod \phi(n)$$

using the extended Euclidean algorithm [19, 31]. Here, e is the public exponent and d is the private exponent. Usually one selects a small public exponent, e.g., $e = 2^{16} + 1$. The modulus n and the public exponent e are published. The value of d and the prime numbers p and q are kept secret. Encryption is performed by computing

$$C = M^e \pmod{n} \ ,$$

where M is the plaintext such that $0 \leq M < n$. The number C is the ciphertext from which the plaintext M can be computed using

$$M = C^d \pmod{n} \ .$$

The correctness of the RSA algorithm follows from Euler's theorem: Let n and a be positive, relatively prime integers. Then

$$a^{\phi(n)} = 1 \pmod{n} \ .$$

Since we have $ed = 1 \bmod \phi(n)$, i.e., $ed = 1 + K\phi(n)$ for some integer K, we can write

$$\begin{aligned} C^d &= (M^e)^d \pmod n \\ &= M^{ed} \pmod n \\ &= M^{1+K\phi(n)} \pmod n \\ &= M \cdot (M^{\phi(n)})^K \pmod n \\ &= M \cdot 1 \pmod n \end{aligned}$$

provided that $\gcd(M, n) = 1$. The exception $\gcd(M, n) > 1$ can be dealt as follows. According to Carmichael's theorem

$$M^{\lambda(n)} = 1 \pmod n$$

where $\lambda(n)$ is Carmichael's function which takes a simple form for $n = pq$, namely,

$$\lambda(pq) = \frac{(p-1)(q-1)}{\gcd(p-1, q-1)} .$$

Note that $\lambda(n)$ is always a proper divisor of $\phi(n)$ when n is the product of distinct odd primes; in this case $\lambda(n)$ is smaller than $\phi(n)$. Now, the relationship between e and d is given by

$$M^{ed} = M \pmod n \text{ if } ed = 1 \pmod{\lambda(n)} .$$

Provided that n is a product of distinct primes, the above holds for all M, thus dealing with the above-mentioned exception $\gcd(M, n) > 1$ in Euler's theorem.

As an example, we construct a simple RSA cryptosystem as follows: Pick $p = 11$ and $q = 13$, and compute

$$\begin{aligned} n &= p \cdot q &&= 11 \cdot 13 &&= 143 , \\ \phi(n) &= (p-1) \cdot (q-1) &&= 10 \cdot 12 &&= 120 . \end{aligned}$$

We can also compute Carmichael's function of n as

$$\lambda(pq) = \frac{(p-1)(q-1)}{\gcd(p-1, q-1)} = \frac{10 \cdot 12}{\gcd(10, 12)} = \frac{120}{2} = 60 .$$

The public exponent e is selected such that $1 < e < \phi(n)$ and

$$\gcd(e, \phi(n)) = \gcd(e, 120) = 1 .$$

For example, $e = 17$ would satisfy this constraint. The private exponent d is computed by

$$\begin{aligned} d &= e^{-1} \pmod{\phi(n)} \\ &= 17^{-1} \pmod{120} \\ &= 113 \end{aligned}$$

which is computed using the extended Euclidean algorithm, or any other algorithm for computing the modular inverse. Thus, the user publishes the public exponent and the modulus: $(e, n) = (13, 143)$, and keeps the following private: $d = 113$, $p = 11$, $q = 13$. A typical encryption/decryption process is executed as follows:

$$
\begin{array}{ll}
\text{Plaintext:} & M = 50 \\
\text{Encryption:} & C := M^e \pmod{n} \\
 & C := 50^{17} \pmod{143} \\
 & C = 85 \\
 & \\
\text{Ciphertext:} & C = 85 \\
\text{Decryption:} & M := M^d \pmod{n} \\
 & M := 85^{113} \pmod{143} \\
 & M = 50
\end{array}
$$

1.2 Exchange of Private Messages

The public-key directory contains the pairs (e, n) for each user. The users wishing to send private messages to one another refer to the directory to obtain these parameters. For example, the directory might be arranged as follows:

User	Public Keys
Alice	(e_a, n_a)
Bob	(e_b, n_b)
Cathy	(e_c, n_c)
...	...

The pair n_a and e_a respectively are the modulus and the public exponent for Alice. As an example, we show how Alice sends her private message M to Bob. In our simple protocol example Alice executes the following steps:

1. Alice locates Bob's name in the directory and obtains his public exponent and the modulus: (e_b, n_b).
2. Alice computes $C := M^{e_b} \pmod{n_b}$.
3. Alice sends C to Bob over the network.
4. Bob receives C.
5. Bob uses his private exponent and the modulus, and computes $M = C^{d_b} \pmod{n_b}$ in order to obtain M.

1.3 Signing Digital Documents

The RSA algorithm provides a procedure for signing a digital document, and verifying whether the signature is indeed authentic. The signing of a digital document is somewhat different from signing a paper document, where the same signature is being produced for all paper documents. A digital signature cannot be a constant; it is a function of the digital

document for which it was produced. After the signature (which is just another piece of digital data) of a digital document is obtained, it is attached to the document for anyone wishing the verify the authenticity of the document and the signature. Here we will briefly illustrate the process of signing using the RSA cryptosystem. Suppose Alice wants to sign a message, and Bob would like to obtain a proof that this message is indeed signed by Alice. First, Alice executes the following steps:

1. Alice takes the message M and computes $S = M^{d_a} \pmod{n_a}$.
2. Alice makes her message M and the signature S available to any party wishing to verify the signature.

Bob executes the following steps in order to verify Alice's signature S on the document M:

1. Bob obtains M and S, and locates Alice's name in the directory and obtains her public exponent and the modulus (e_a, n_a).
2. Bob computes $M' = S^{e_a} \pmod{n_a}$.
3. If $M' = M$ then the signature is verified. Otherwise, either the original message M or the signature S is modified, thus, the signature is not valid.

We note that the protocol examples given here for illustration purposes only — they are simple 'textbook' protocols; in practice, the protocols are somewhat more complicated. For example, secret-key cryptographic techniques may also be used for sending private messages. Also, signing is applied to messages of arbitrary length. The signature is often computed by first computing a hash value of the long message and then signing this hash value. We refer the reader to the report [42] and Public Key Cryptography Standards [43] published by RSA Data Security, Inc., for answers to certain questions on these issues.

1.4 Computation of Modular Exponentiation

Once an RSA cryptosystem is set up, i.e., the modulus and the private and public exponents are determined and the public components have been published, the senders as well as the recipients perform a single operation for signing, verification, encryption, and decryption. The RSA algorithm in this respect is one of the simplest cryptosystems. The operation required is the computation of $M^e \pmod{n}$, i.e., the modular exponentiation. The modular exponentiation operation is a common operation for scrambling; it is used in several cryptosystems. For example, the Diffie-Hellman key exchange scheme requires modular exponentiation [8]. Furthermore, the ElGamal signature scheme [13] and the recently proposed Digital Signature Standard (DSS) of the National Institute for Standards and Technology [34] also require the computation of modular exponentiation. However, we note that the exponentiation process in a cryptosystem based on the discrete logarithm problem is slightly different: The base (M) and the modulus (n) are known in advance. This allows some precomputation since

powers of the base can be precomputed and saved [6]. In the exponentiation process for the RSA algorithm, we know the exponent (e) and the modulus (n) in advance but not the base; thus, such optimizations are not likely to be applicable. The emphasis of this report is on the RSA cryptosystem as the title suggests.

In the following chapters we will review techniques for implementation of modular exponentiation operation on general-purpose computers, e.g., personal computers, microprocessors, microcontrollers, signal processors, workstations, and mainframe computers. This report does not include any *actual* code; it covers mathematical and algorithmic aspects of the *software* implementations of the RSA algorithm. There also exist hardware structures for performing the modular multiplication and exponentiations, for example, see [40, 28, 46, 15, 24, 25, 26, 50]. A brief review of the hardware implementations can be found in [5].

Chapter 2

Modular Exponentiation

2.1 Modular Exponentiation

The first rule of modular exponentiation is that we do *not* compute

$$C := M^e \pmod{n}$$

by first exponentiating

$$M^e$$

and then performing a division to obtain the remainder

$$C := (M^e) \% n .$$

The temporary results *must* be reduced modulo n at each step of the exponentiation. This is because the space requirement of the binary number M^e is enormous. Assuming, M and e have 256 bits each, we need

$$\log_2(M^e) = e \cdot \log_2(M) \approx 2^{256} \cdot 256 = 2^{264} \approx 10^{80}$$

bits in order to store M^e. This number is approximately equal to the number of particles in the universe [1]; we have no way of storing it. In order to compute the bit capacity of all computers in the world, we can make a generous assumption that there are 512 million computers, each of which has 512 MBytes of memory. Thus, the total number of bits available would be

$$512 \cdot 2^{20} \cdot 512 \cdot 2^{20} \cdot 8 = 2^{61} \approx 10^{18} ,$$

which is only enough to store M^e when M and e are 55 bits.

2.2 Exponentiation

We raise the following question: *How many modular multiplications are needed to compute* $M^e \bmod n$ *?* A naive way of computing $C = M^e \pmod{n}$ is to start with $C := M$

(mod n) and keep performing the modular multiplication operations

$$C := C \cdot M \pmod{n}$$

until $C = M^e \pmod{n}$ is obtained. The naive method requires $e-1$ modular multiplications to compute $C := M^e \pmod{n}$, which would be prohibitive for large e. For example, if we need to compute $M^{15} \pmod{n}$, this method computes all powers of M until 15:

$$M \to M^2 \to M^3 \to M^4 \to M^5 \to M^6 \to M^7 \to \cdots \to M^{15}$$

which requires 14 multiplications. However, not all powers of M need to be computed in order to obtain M^{15}. Here is a faster method of computing M^{15}:

$$M \to M^2 \to M^3 \to M^6 \to M^7 \to M^{14} \to M^{15}$$

which requires 6 multiplications. The method by which M^{15} is computed is not specific for certain exponents; it can be used to compute M^e for any e. The algorithm is called the *binary method* or *square and multiply* method, and dates back to antiquity.

2.3 The Binary Method

The binary method scans the bits of the exponent either from left to right or from right to left. A squaring is performed at each step, and depending on the scanned bit value, a subsequent multiplication is performed. We describe the left-to-right binary method below. The right-to-left algorithm requires one extra variable to keep the powers of M. The reader is referred to Section 4.6.3 of Knuth's book [19] for more information. Let k be the number of bits of e, i.e., $k = 1 + \lfloor \log_2 e \rfloor$, and the binary expansion of e be given by

$$e = (e_{k-1}e_{k-2}\cdots e_1e_0) = \sum_{i=0}^{k-1} e_i 2^i$$

for $e_i \in \{0,1\}$. The binary method for computing $C = M^e \pmod{n}$ is given below:

The Binary Method
Input: M, e, n.
Output: $C = M^e \bmod n$.
1. **if** $e_{k-1} = 1$ **then** $C := M$ **else** $C := 1$
2. **for** $i = k-2$ **downto** 0
   2a. $C := C \cdot C \pmod{n}$
   2b. **if** $e_i = 1$ **then** $C := C \cdot M \pmod{n}$
3. **return** C

As an example, let $e = 250 = (11111010)$, which implies $k = 8$. Initially, we take $C := M$ since $e_{k-1} = e_7 = 1$. The binary method proceeds as follows:

i	e_i	Step 2a	Step 2b
6	1	$(M)^2 = M^2$	$M^2 \cdot M = M^3$
5	1	$(M^3)^2 = M^6$	$M^6 \cdot M = M^7$
4	1	$(M^7)^2 = M^{14}$	$M^{14} \cdot M = M^{15}$
3	1	$(M^{15})^2 = M^{30}$	$M^{30} \cdot M = M^{31}$
2	0	$(M^{31})^2 = M^{62}$	M^{62}
1	1	$(M^{62})^2 = M^{124}$	$M^{124} \cdot M = M^{125}$
0	0	$(M^{125})^2 = M^{250}$	M^{250}

The number of modular multiplications required by the binary method for computing M^{250} is found to be $7+5=12$. For an arbitrary k-bit number e with $e_{k-1}=1$, the binary method requires:

- Squarings (Step 2a): $k-1$ where k is the number of bits in the binary expansion of e.
- Multiplications (Step 2b): $H(e)-1$ where $H(e)$ is the Hamming weight (the number of 1s in the binary expansion) of e.

Assuming $e > 0$, we have $0 \leq H(e)-1 \leq k-1$. Thus, the total number of multiplications is found as:

$$\begin{aligned} \text{Maximum:} \quad (k-1)+(k-1) &= 2(k-1) \ , \\ \text{Minimum:} \quad (k-1)+0 &= k-1 \ , \\ \text{Average:} \quad (k-1)+\tfrac{1}{2}(k-1) &= \tfrac{3}{2}(k-1) \ , \end{aligned}$$

where we assume that $e_{k-1}=1$.

2.4 The m-ary Method

The binary method can be generalized by scanning the bits of e

- 2 at a time: the quaternary method, or
- 3 at a time: the octal method, etc.

More generally,

- $\log_2 m$ at a time: the m-ary method.

The m-ary method is based on m-ary expansion of the exponent. The digits of e are then scanned and squarings (powerings) and subsequent multiplications are performed accordingly. The method was described in Knuth's book [19]. When m is a power of 2, the implementation of the m-ary method is rather simple, since M^e is computed by grouping

the bits of the binary expansion of the exponent e. Let $e = (e_{k-1}e_{k-2}\cdots e_1e_0)$ be the binary expansion of the exponent. This representation of e is partitioned into s blocks of length r each for $sr = k$. If r does not divide k, the exponent is padded with at most $r-1$ 0s. We define

$$F_i = (e_{ir+r-1}e_{ir+r-2}\cdots e_{ir}) = \sum_{j=0}^{r-1} e_{ir+j}2^j \ .$$

Note that $0 \le F_i \le m-1$ and $e = \sum_{i=0}^{s-1} F_i 2^{ir}$. The m-ary method first computes the values of $M^w \pmod n$ for $w = 2, 3, \ldots, m-1$. Then the bits of e are scanned r bits at a time from the most significant to the least significant. At each step the partial result is raised to the 2^r power and multiplied y M^{F_i} modulo n where F_i is the (nonzero) value of the current bit section.

The m-ary Method
Input: M, e, n.
Output: $C = M^e \bmod n$.

1. Compute and store $M^w \pmod n$ for all $w = 2, 3, 4, \ldots, m-1$.
2. Decompose e into r-bit words F_i for $i = 0, 1, 2, \ldots, s-1$.
3. $C := M^{F_{s-1}} \pmod n$
4. **for** $i = s-2$ **downto** 0
   4a. $C := C^{2^r} \pmod n$
   4b. **if** $F_i \neq 0$ **then** $C := C \cdot M^{F_i} \pmod n$
5. **return** C

2.4.1 The Quaternary Method

We first consider the quaternary method. Since the bits of e are scanned two at a time, the possible digit values are $(00) = 0$, $(01) = 1$, $(10) = 2$, and $(11) = 3$. The multiplication step (Step 4b) may require the values M^0, M^1, M^2, and M^3. Thus, we need to perform some preprocessing to obtain M^2 and M^3. As an example, let $e = 250$ and partition the bits of e in groups of two bits as

$$e = 250 = \underline{11}\ \underline{11}\ \underline{10}\ \underline{10} \ .$$

Here, we have $s = 4$ (the number of groups $s = k/r = 8/2 = 4$). During the preprocessing step, we compute:

bits	w	M^w
00	0	1
01	1	M
10	2	$M \cdot M = M^2$
11	3	$M^2 \cdot M = M^3$

The quaternary method then assigns $C := M^{F_3} = M^3 \pmod n$, and proceeds to compute $M^{250} \pmod n$ as follows:

i	F_i	Step 4a	Step 4b
2	11	$(M^3)^4 = M^{12}$	$M^{12} \cdot M^3 = M^{15}$
1	10	$(M^{15})^4 = M^{60}$	$M^{60} \cdot M^2 = M^{62}$
0	10	$(M^{62})^4 = M^{248}$	$M^{248} \cdot M^2 = M^{250}$

The number of modular multiplications required by the quaternary method for computing $M^{250} \pmod{n}$ is found as $2 + 6 + 3 = 11$.

2.4.2 The Octal Method

The octal method partitions the bits of the exponent in groups of 3 bits. For example, $e = 250$ is partitioned as

$$e = 250 = \underline{011}\ \underline{111}\ \underline{010}\ ,$$

by padding a zero to the left, giving $s = k/r = 9/3 = 3$. During the preprocessing step we compute $M^w \pmod{n}$ for all $w = 2, 3, 4, 5, 6, 7$.

bits	w	M^w
000	0	1
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M^2 \cdot M = M^3$
100	4	$M^3 \cdot M = M^4$
101	5	$M^4 \cdot M = M^5$
110	6	$M^5 \cdot M = M^6$
111	7	$M^6 \cdot M = M^7$

The octal method then assigns $C := M^{F_2} = M^3 \pmod{n}$, and proceeds to compute $M^{250} \pmod{n}$ as follows:

i	F_i	Step 4a	Step 4b
1	111	$(M^3)^8 = M^{24}$	$M^{24} \cdot M^7 = M^{31}$
0	010	$(M^{31})^8 = M^{248}$	$M^{248} \cdot M^2 = M^{250}$

The computation of $M^{250} \pmod{n}$ by the octal method requires a total of $6 + 6 + 2 = 14$ modular multiplications. However, notice that, even though we have computed $M^w \pmod{n}$ for all $w = 2, 3, 4, 5, 6, 7$, we have not used *all* of them. Thus, we can slightly modify Step 1 of the m-ary method and precompute $M^w \pmod{n}$ for only those w which appear in the partitioned binary expansion of e. For example, for $e = 250$, the partitioned bit values are $(011) = 3$, $(111) = 7$, and $(010) = 2$. We can compute these powers using only 4 multiplications:

bits	w	M^w
000	0	1
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M^2 \cdot M = M^3$
100	4	$M^3 \cdot M = M^4$
111	7	$M^4 \cdot M^3 = M^7$

This gives the total number of multiplications required by the octal method for computing $M^{250} \pmod{n}$ as $4+6+2 = 12$. The method of computing $M^e \pmod{n}$ by precomputing $M^w \pmod{n}$ for only those w which appear in the partitioning of the exponent is termed a data-dependent or an adaptive algorithm. In the following section, we will explore methods of this kind which try to reduce the number of multiplications by making use of the properties of the given e. In general, we will probably have to compute $M^w \pmod{n}$ for all $w = 2, 3, \ldots, 2^r - 1$. This will be more of the case when k is very large. We summarize the average number of multiplications and squarings required by the m-ary method assuming $2^r = m$ and $\frac{k}{r}$ is an integer.

- Preprocessing Multiplications (Step 1): $m - 2 = 2^r - 2$
- Squarings (Step 4a): $(\frac{k}{r} - 1) \cdot r = k - r$
- Multiplications (Step 4b): $(\frac{k}{r} - 1)(1 - \frac{1}{m}) = (\frac{k}{r} - 1)(1 - 2^{-r})$

Thus, in general, the m-ary method requires

$$2^r - 2 + k - r + \left(\frac{k}{r} - 1\right)\left(1 - 2^{-r}\right)$$

multiplications plus squarings on the average. The average number of multiplications for the binary method can be found simply by substituting $r = 1$ and $m = 2$ in the above, which gives $\frac{3}{2}(k-1)$. Also note that there exists an optimal $r = r^*$ for each k such that the average number of multiplications required by the m-ary method is minimum. The optimal values of r can be found by enumeration [21]. In the following we tabulate the average values of multiplications plus squarings required by the binary method and the m-ary method with the optimal values of r.

k	binary	m-ary	r^*	Savings %
8	11	10	2	9.1
16	23	21	2	8.6
32	47	43	2,3	8.5
64	95	85	3	10.5
128	191	167	3,4	12.6
256	383	325	4	15.1
512	767	635	5	17.2
1024	1535	1246	5	18.8
2048	3071	2439	6	20.6

The asymptotic value of savings offered by the m-ary method is equal to 33 %. In order to prove this statement, we compute the limit of the ratio

$$\lim_{k\to\infty} \frac{2^r - 2 + k - r + (\frac{k}{r} - 1)(1 - 2^{-r})}{\frac{3}{2}(k-1)} = \frac{2}{3}\left(1 + \frac{1 - 2^{-r}}{r}\right) \approx \frac{2}{3} .$$

2.5 The Adaptive m-ary Methods

The adaptive methods are those which form their method of computation according to the input data. In the case of exponentiation, an adaptive algorithm will modify its structure according to the exponent e, once it is supplied. As we have pointed out earlier, the number of preprocessing multiplications can be reduced if the partitioned binary expansion of e do not contain all possible bit-section values w. However, there are also adaptive algorithms which partition the exponent into a series of zero and nonzero words in order to decrease the number multiplications required in Step 4b of the m-ary method. In the following we introduce these methods, and give the required number of multiplications and squarings.

2.5.1 Reducing Preprocessing Multiplications

We have already briefly introduced this method. Once the binary expansion of the exponent is obtained, we partition this number into groups of d bits each. We then precompute and obtain $M^w \pmod n$ only for those w which appear in the binary expansion. Consider the following exponent for $k = 16$ and $d = 4$

$$\underline{1011}\ \underline{0011}\ \underline{0111}\ \underline{1000}$$

which implies that we need to compute $M^w \pmod n$ for only $w = 3, 7, 8, 11$. The exponent values $w = 3, 7, 8, 11$ can be sequentially obtained as follows:

$$\begin{aligned} M^2 &= M \cdot M \\ M^3 &= M^2 \cdot M \\ M^4 &= M^2 \cdot M^2 \\ M^7 &= M^3 \cdot M^4 \\ M^8 &= M^4 \cdot M^4 \\ M^{11} &= M^8 \cdot M^3 \end{aligned}$$

which requires 6 multiplications. The m-ary method that disregards the necessary exponent values and computes all of them would require $16 - 2 = 14$ preprocessing multiplications. The number of multiplications that can be saved is upper-bounded by $m - 2 = 2^d - 2$, which is the case when all partitioned exponent values are equal to 1, e.g., when

$$\underline{0001}\ \underline{0001}\ \underline{0001}\ \underline{0001} \quad .$$

This implies that we do not precompute anything, just use M. This happens quite rarely. In general, we have to compute $M^w \pmod n$ for all $w = w_0, w_1, \ldots, w_{p-1}$. If the span of the set $\{w_i \mid i = 0, 1, \ldots, p-1\}$ is the values $2, 3, \ldots, 2^d - 1$, then there is no savings. We perform $2^d - 2$ multiplications and obtain all of these values. However, if the span is a subset (especially a small subset) of the values $2, 3, \ldots, 2^d - 1$, then some savings can be achieved if we can compute w_i for $i = 0, 1, \ldots, p-1$ using much fewer than $2^d - 2$ multiplications. An algorithm for computing any given p exponent values is called *a vectorial addition chain*, and in the case of $p = 1$, *an addition chain*. Unfortunately, the problem of obtaining an addition chain of minimal length is an NP-complete problem [9]. We will elaborate on addition and vectorial addition chains in the last section of this chapter.

2.5.2 The Sliding Window Techniques

The m-ary method decomposes the bits of the exponent into d-bit words. The probability of a word of length d being zero is equal to 2^{-d}, assuming that the 0 and 1 bits are produced with equal probability. In Step 4b of the m-ary method, we skip a multiplication whenever the current word is equal to zero. Thus, as d grows larger, the probability that we have to perform a multiplication operation in Step 4a becomes larger. However, the total number of multiplications increases as d decreases. The sliding window algorithms provide a compromise by allowing zero and nonzero words of variable length; this strategy aims to increase the average number of zero words, while using relatively large values of d.

A sliding window exponentiation algorithm first decomposes e into zero and nonzero words (*windows*) F_i of length $L(F_i)$. The number of windows p may not be equal to k/d. In general, it is also not required that the length of the windows be equal. We take d to be the length of the longest window, i.e., $d = \max(L(F_i))$ for $i = 0, 1, \ldots, k-1$. Furthermore, if F_i is a nonzero window, then the least significant bit of F_i must be equal to 1. This is because we partition the exponent starting from the least significant bit, and there is no point in starting a nonzero window with a zero bit. Consequently, the number of preprocessing multiplications (Step 1) are nearly halved, since x^w are computed for odd w only.

The Sliding Window Method
Input: M, e, n.
Output: $C = M^e \pmod n$.
1. Compute and store $M^w \pmod n$ for all $w = 3, 5, 7, \ldots, 2^d - 1$.
2. Decompose e into zero and nonzero windows F_i of length $L(F_i)$ for $i = 0, 1, 2, \ldots, p-1$.
3. $C := M^{F_{k-1}} \pmod n$
4. **for** $i = p-2$ **downto** 0
   4a. $C := C^{2^{L(F_i)}} \pmod n$
   4b. **if** $F_i \neq 0$ **then** $C := C \cdot M^{F_i} \pmod n$
5. **return** C

Two sliding window partitioning strategies have been proposed [19, 4]. These methods differ

in whether the length of a nonzero window must be a constant (= d), or can be variable (however, $\leq d$). In the following sections, we give algorithmic descriptions of these two partitioning strategies.

2.5.3 Constant Length Nonzero Windows

The constant length nonzero window (CLNW) partitioning algorithm is due to Knuth [19]. The algorithm scans the bits of the exponent from the least significant to the most significant. At any step, the algorithm is either forming a zero window (ZW) or a nonzero window (NW). The algorithm is described below:

ZW: Check the incoming single bit: if it is a 0 then stay in ZW; else go to NW.

NW: Stay in NW until all d bits are collected. Then check the incoming single bit: if it is a 0 then go to ZW; else go to NW.

Notice that while in NW, we distinguish between staying in NW and going to NW. The former means that we continue to form the same nonzero window, while the latter implies the beginning of a new nonzero window. The CLNW partitioning strategy produces zero windows of arbitrary length, and nonzero windows of length d. There cannot be two adjacent zero windows; they are necessarily concatenated, however, two nonzero windows may be adjacent. For example, for $d = 3$, we partition $e = 3665 = (111001010001)$ as

$$e = \underline{111}\ 00\ \underline{101}\ 0\ \underline{001}\ .$$

The CLNW sliding window algorithm first performs the preprocessing multiplications and obtains $M^w \pmod{n}$ for $w = 3, 5, 7$.

bits	w	M^w
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M \cdot M^2 = M^3$
101	5	$M^3 \cdot M^2 = M^5$
111	7	$M^5 \cdot M^2 = M^7$

The algorithm assigns $C = M^{F_4} = M^7 \pmod{n}$, and then proceeds to compute $M^{3665} \pmod{n}$ as follows:

i	F_i	$L(F_i)$	Step 4a	Step 4b
3	00	2	$(M^7)^4 = M^{28}$	M^{28}
2	101	3	$(M^{28})^8 = M^{224}$	$M^{224} \cdot M^5 = M^{229}$
1	0	1	$(M^{229})^2 = M^{458}$	M^{458}
0	001	3	$(M^{458})^8 = M^{3664}$	$M^{3664} \cdot M = M^{3665}$

Thus, a total of $4+9+2=15$ modular multiplications are performed. The average number of multiplications can be found by modeling the partitioning process as a Markov chain. The details of this analysis are given in [23]. In following table, we tabulate the average number of multiplications for the m-ary and the CLNW sliding window methods. The column for the m-ary method contains the optimal values d^* for each k. As expected, there exists an optimal value of d for each k for the CLNW sliding window algorithm. These optimal values are also included in the table. The last column of the table contains the percentage difference in the average number of multiplications. The CLNW partitioning strategy reduces the average number of multiplications by 3–7 % for $128 \leq k \leq 2048$. The overhead of the partitioning is negligible; the number of bit operations required to obtain the partitioning is proportional to k.

	m-ary		CLNW		$(T-T_1)/T$
k	d^*	T	d^*	T_1	%
128	4	168	4	156	7.14
256	4	326	5	308	5.52
512	5	636	5	607	4.56
768	5	941	6	903	4.04
1024	5	1247	6	1195	4.17
1280	6	1546	6	1488	3.75
1536	6	1844	6	1780	3.47
1792	6	2142	7	2072	3.27
2048	6	2440	7	2360	3.28

2.5.4 Variable Length Nonzero Windows

The CLNW partitioning strategy starts a nonzero window when a 1 is encountered. Although the incoming $d-1$ bits may all be zero, the algorithm continues to append them to the current nonzero window. For example, for $d=3$, the exponent $e=(111001010001)$ was partitioned as

$$e = \underline{111}\ 00\ \underline{101}\ 0\ \underline{001}\ .$$

However, if we allow variable length nonzero windows, we can partition this number as

$$e = \underline{111}\ 00\ \underline{101}\ 000\ \underline{1}\ .$$

We will show that this strategy further decreases the average number of nonzero windows. The variable length nonzero window (VLNW) partitioning strategy was proposed by Bos and Coster in [4]. The strategy requires that during the formation of a nonzero window (NW), we switch to ZW when the remaining bits are all zero. The VLNW partitioning strategy has two integer parameters:

- d : maximum nonzero window length,

- q : minimum number of zeros required to switch to ZW.

The algorithm proceeds as follows:

ZW: Check the incoming single bit: if it is zero then stay in ZW; else go to NW.

NW: Check the incoming q bits: if they are all zero then go to ZW; else stay in NW. Let $d = lq + r + 1$ where $1 < r \leq q$. Stay in NW until $lq + 1$ bits are received. At the last step, the number of incoming bits will be equal to r. If these r bits are all zero then go to ZW; else stay in NW. After all d bits are collected, check the incoming single bit: if it is zero then go to ZW; else go to NW.

The VLNW partitioning produces nonzero windows which start with a 1 and end with a 1. Two nonzero windows may be adjacent; however, the one in the least significant position will necessarily have d bits. Two zero windows will not be adjacent since they will be concatenated. For example, let $d = 5$ and $q = 2$, then $5 = 1 + 1 \cdot 2 + 2$, thus $l = 1$ and $r = 2$. The following illustrates the partitioning of a long exponent according to these parameters:

$$\underline{101}\ 0\ \underline{11101}\ 00\ \underline{101}\ \underline{10111}\ 000000\ \underline{1}\ 00\ \underline{111}\ 000\ \underline{1011}\ .$$

Also, let $d = 10$ and $q = 4$, which implies $l = 2$ and $r = 1$. A partitioning example is illustrated below:

$$\underline{1011011}\ 0000\ \underline{11}\ 0000\ \underline{11110111}\ 00\ \underline{1111110101}\ 0000\ \underline{11011}\ .$$

In order to compute the average number of multiplications, the VLNW partitioning process, like the CLNW process, can be modeled using a Markov chain. This analysis was performed in [23], and the average number of multiplications have been calculated for $128 \leq k \leq 2048$. In the following table, we tabulate these values together with the optimal values of d and q, and compare them to those of the m-ary method. Experiments indicate that the best values of q are between 1 and 3 for $128 \leq k \leq 2048$ and $4 \leq d \leq 8$. The VLNW algorithm requires 5–8 % fewer multiplications than the m-ary method.

	m-ary		VLNW				$(T_2-T)/T_2$
					T_2/k		for q^*
k	d^*	T/k	d^*	$q=1$	$q=2$	$q=3$	%
128	4	1.305	4	1.204	1.203	1.228	7.82
256	4	1.270	4	1.184	1.185	1.212	6.77
512	5	1.241	5	1.163	1.175	1.162	6.37
768	5	1.225	5	1.155	1.167	1.154	5.80
1024	5	1.217	6	1.148	1.146	1.157	5.83
1280	6	1.207	6	1.142	1.140	1.152	5.55
1536	6	1.200	6	1.138	1.136	1.148	5.33
1792	6	1.195	6	1.136	1.134	1.146	5.10
2048	6	1.191	6	1.134	1.132	1.144	4.95

The sliding window algorithms are easy to program, introducing negligible overhead. The reduction in terms of the number of multiplications is notable, for example, for $n = 512$, the m-ary method requires 636 multiplications whereas the CLNW and VLNW sliding window algorithms require 607 and 595 multiplications, respectively. In Figure 2.1, we plot the scaled average number of multiplications T/k, i.e., the average number of multiplications T divided by the total number of bits k, for the m-ary and the sliding window algorithms as a function of $n = 128, 256, \ldots, 2048$.

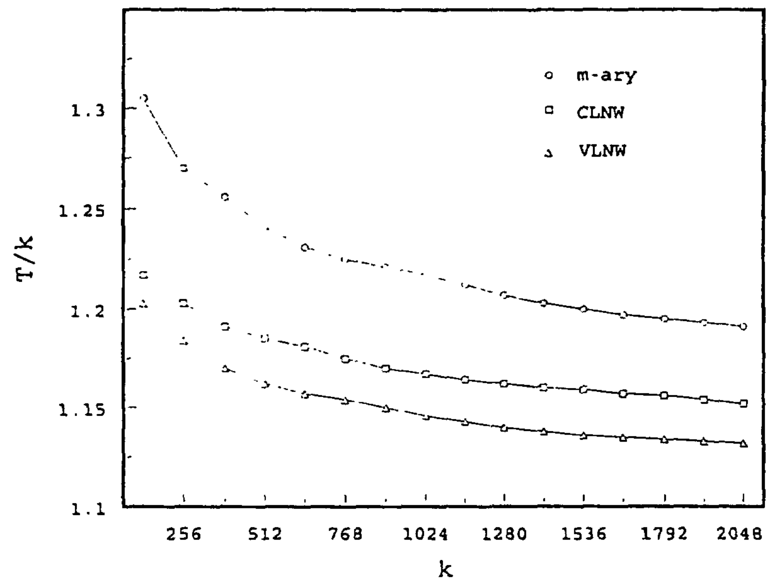


Figure 2.1: The values of T/k for the m-ary and the sliding window algorithms.

2.6 The Factor Method

The factor method is given by Knuth [19]. It is based on factorization of the exponent $e = rs$ where r is the smallest prime factor of e and $s > 1$. We compute M^e by first computing M^r and then raising this value to the sth power:

$$\begin{aligned} C_1 &= M^r , \\ C_2 &= C_1^s = M^{rs} = M^e . \end{aligned}$$

If e is prime, then we first compute M^{e-1} then multiply this quantity by M. The algorithm is recursive, e.g., in order to compute M^r, we factor $r = r_1 \cdot r_2$ such that r_1 is the smallest prime factor of r and $r_2 > 1$. This process continues until the exponent value required is equal to 2. As an example, we illustrate the computation of M^e for $e = 55 = 5 \cdot 11$ in the following:

$$
\begin{array}{ll}
\text{Compute:} & M \rightarrow M^2 \rightarrow M^4 \rightarrow M^5 \\
\text{Assign:} & a := M^5 \\
\text{Compute:} & a \rightarrow a^2 \\
\text{Assign:} & b := a^2 \\
\text{Compute:} & b \rightarrow b^2 \rightarrow b^4 \rightarrow b^5 \\
\text{Compute:} & b^5 \rightarrow b^5 a = M^{55}
\end{array}
$$

The factor method requires 8 multiplications for computing M^{55}. The binary method, on the other hand, requires 9 multiplications since $e = 55 = (110111)$ implies 5 squarings (Step 2a) and 4 multiplications (Step 2b).

Unfortunately, the factor method requires factorization of the exponent, which would be very difficult for large numbers. However, this method could still be of use for the RSA cryptosystem whenever the exponent value is small. It may also be useful if the exponent is constructed carefully, i.e., in a way to allow easy factorization.

2.7 The Power Tree Method

The power tree method is also due to Knuth [19]. This algorithm constructs a tree according to a heuristic. The nodes of the tree are labeled with positive integers starting from 1. The root of the tree receives 1. Suppose that the tree is constructed down to the kth level. Consider the node e of the kth level, from left to right. Construct the $(k+1)$st level by attaching below node e the nodes

$$e + a_1, e + a_2, e + a_3, \ldots, e + a_k$$

where $a_1, a_2, a_3, \ldots, a_k$ is the path from the root of the tree to e. (Note: $a_1 = 1$ and $a_k = e$.) In this process, we discard any duplicates that have already appeared in the tree. The power tree down to 5 levels is given in Figure 2.2.

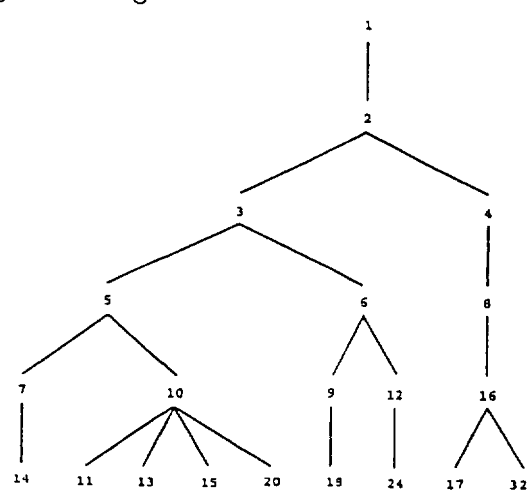


Figure 2.2: The Power Tree .

In order to compute M^e, we locate e in the power tree. The sequence of exponents that occur in the computation of M^e is found on the path from the root to e. For example, the computation of M^{18} requires 5 multiplications:

$$M \to M^2 \to M^3 \to M^6 \to M^9 \to M^{18}$$

For certain exponent values of e, the power tree method requires fewer number of multiplications, e.g., the computation of M^{23} by the power tree method requires 6 multiplications:

$$M \to M^2 \to M^3 \to M^5 \to M^{10} \to M^{13} \to M^{23}$$

However, since $23 = (10111)$, the binary method requires $4 + 3 = 7$ multiplications:

$$M \to M^2 \to M^4 \to M^5 \to M^{10} \to M^{11} \to M^{22} \to M^{23}$$

Also, since $23 - 1 = 22 = 2 \cdot 11$, the factor method requires $1 + 5 + 1 = 7$ multiplications:

$$M \to M^2 \to M^4 \to M^8 \to M^{16} \to M^{20} \to M^{22} \to M^{23}$$

Knuth gives another variation of the power tree heuristics in Problem 6 in page 463 [19]. The power tree method is also applicable for small exponents since the tree needs to be "saved".

2.8 Addition Chains

Consider a sequence of integers

$$a_0, a_1, a_2, \ldots, a_r$$

with $a_0 = 1$ and $a_r = e$. If the sequence is constructed in such a way that for all k there exist indices $i, j < k$ such that

$$a_k = a_i + a_j \ ,$$

then the sequence is an addition chain for e. The length of the chain is equal to r. An addition chain for a given exponent e is an algorithm for computing M^e. We start with M^1, and proceed to compute M^{a_k} using the two previously computed values M^{a_i} and M^{a_j} as $M^{a_k} = M^{a_i} \cdot M^{a_j}$. The number of multiplications required is equal to r which is the length of the addition chain. The algorithms we have so far introduced, namely, the binary method, the m-ary method, the sliding window method, the factor and the power tree methods are in fact methods of generating addition chains for the given exponent value e. Consider for example $e = 55$, the addition chains generated by some of these algorithms are given below:

binary:	1	2	3	6	12	13	26	27	54	55	
quaternary:	1	2	3	6	12	13	26	52	55		
octal:	1	2	3	4	5	6	7	12	24	48	55
factor:	1	2	4	5	10	20	40	50	55		
power tree:	1	2	3	5	10	11	22	44	55		

Given the positive integer e, the computation of the *shortest* addition chain for e is established to be an NP-complete problem [9]. This implies that we have to compute all possible chains leading to e in order to obtain the shortest one. However, since the first introduction of the shortest addition chain problem by Scholz [19] in 1937, several properties of the addition chains have been established:

- The upper bound on the length of the shortest addition chain for e is equal to: $\lfloor \log_2 e \rfloor + H(e) - 1$ where $H(e)$ is the Hamming weight of e. This follows from the binary method. In the worst case, we can use the binary method to compute M^e using at most $\lfloor \log_2 e \rfloor + H(e) - 1$ multiplications.

- The lower bound was established by Schönhage [44]: $\log_2 e + \log_2 H(e) - 2.13$. Thus, no addition chain for e can be shorter than $\log_2 e + \log_2 H(e) - 2.13$.

The previously given algorithms for computing M^e are all *heuristics* for generating short addition chains. We call these algorithms heuristics because they do not guarantee minimality. Statistical methods, such as simulated annealing, can be used to produce short addition chains for certain exponents. Certain heuristics for obtaining short addition chains are discussed in [4, 52].

2.9 Vectorial Addition Chains

Another related problem (which we have briefly mentioned in Section 2.5.1) is the generation of *vectorial* addition chains. A vectorial addition chain of a given vector of integer components is the list of vectors with the following properties:

- The initial vectors are the unit vectors $[1,0,\ldots,0], [0,1,0,\ldots,0], \ldots, [0,\ldots,0,1]$.

- Each vector is the sum of two earlier vectors.

- The last vector is equal to the given vector.

For example, given the vector $[7, 15, 23]$, we obtain a vectorial addition chain as

$$\begin{array}{llllllllll} [1,0,0] \\ [0,1,0] & [0,1,1] & [1,1,1] & [0,1,2] & [1,2,3] & [1,3,5] & [2,4,6] & [3,7,11] & [4,8,12] & [7,15,23] \\ [0,0,1] \end{array}$$

which is of length 9. Short vectorial addition chains can be used to efficiently compute M^{w_i} for several integers w_i. This problem arises in conjunction with reducing the preprocessing multiplications in adaptive m-ary methods and as well as in the sliding window technique (refer to Section 2.5). If the exponent values appear in the partitioning of the binary expansion of e are just 7, 15, and 23, then the above vectorial addition chain can be used for

obtaining these exponent values. This is achieved by noting a one-to-one correspondence between the addition sequences and the vectorial addition chains. This result was established by Olivos [35] who proved that the complexity of the computation of the multinomial

$$x_1^{n_1} x_2^{n_2} \cdots x_i^{n_i}$$

is the same as the simultaneous computation of the monomials

$$x^{n_1}, x^{n_2}, \ldots, x^{n_i} .$$

An addition sequence is simply an addition chain where the i requested numbers $n_1, n_2, \ldots, n_i$ occur somewhere in the chain [53]. Using the Olivos algorithm, we convert the above vectorial addition chain to the addition sequence with the requested numbers 7, 15, and 23 as

$$1 \quad 2 \quad 3 \quad 4 \quad 7 \quad 8 \quad 15 \quad 23$$

which is of length 7. In general an addition sequence of length r and i requested numbers can be converted to a vectorial addition sequence of length $r + i - 1$ with dimension i.

2.10 Recoding Methods

In this section we discuss exponentiation algorithms which are intrinsically different from the ones we have so far studied. The property of these algorithms is that they require the inverse of M modulo n in order to efficiently compute $M^e \pmod n$. It is established that $k - 1$ is a lower bound for the number of squaring operations required for computing M^e where k is the number of bits in e. However, it is possible to reduce the number of consequent multiplications using a *recoding* of the the exponent [17, 33, 11, 21]. The recoding techniques use the identity

$$2^{i+j-1} + 2^{i+j-2} + \cdots + 2^i = 2^{i+j} - 2^i$$

to collapse a block of 1s in order to obtain a *sparse* representation of the exponent. Thus, a redundant signed-digit representation of the exponent using the digits $\{0, 1, -1\}$ will be obtained. For example, (011110) can be recoded as

$$\begin{aligned} (011110) &= 2^4 + 2^3 + 2^2 + 2^1 \\ (1000\bar{1}0) &= 2^5 - 2^1 . \end{aligned}$$

Once a recoding of the exponent is obtained, we can use the binary method (or, the m-ary method) to compute $M^e \pmod n$ provided that $M^{-1} \pmod n$ is supplied along with M. For example, the recoding binary method is given below:

The Recoding Binary Method
Input: M, M^{-1}, e, n.
Output: $C = M^e \bmod n$.

0. Obtain a signed digit representation f of e.
1. **if** $f_k = 1$ **then** $C := M$ else $C := 1$
2. **for** $i = k - 1$ **downto** 0
   2a. $C := C \cdot C \pmod{n}$
   2b. **if** $f_i = 1$ **then** $C := C \cdot M \pmod{n}$
   else if $f_i = \bar{1}$ **then** $C := C \cdot M^{-1} \pmod{n}$
3. **return** C

Note that even though the number of bits of e is equal to k, the number of bits in the the recoded exponent f can be $k + 1$, for example, (111) is recoded as $(100\bar{1})$. Thus, the recoding binary algorithm starts from the bit position k in order to compute $M^e \pmod{n}$ by computing $M^f \pmod{n}$ where f is the $(k+1)$-bit recoded exponent such that $f = e$. We give an example of exponentiation using the recoding binary method. Let $e = 119 = (1110111)$. The (nonrecoding) binary method requires $6 + 5 = 11$ multiplications in order to compute $M^{119} \pmod{n}$. In the recoding binary method, we first obtain a sparse signed-digit representation of 119. We will shortly introduce techniques for obtaining such recodings. For now, it is easy to verify the following:

$$\begin{array}{rcll} \text{Exponent:} & 119 & = & 01110111\ , \\ \text{Recoded Exponent:} & 119 & = & 1000\bar{1}00\bar{1}\ . \end{array}$$

The recoding binary method then computes $M^{119} \pmod{n}$ as follows:

f_i	Step 2a	Step 2b
1	M	M
0	$(M)^2 = M^2$	M^2
0	$(M^2)^2 = M^4$	M^4
0	$(M^4)^2 = M^8$	M^8
$\bar{1}$	$(M^8)^2 = M^{16}$	$M^{16} \cdot M^{-1} = M^{15}$
0	$(M^{15})^2 = M^{30}$	M^{30}
0	$(M^{30})^2 = M^{60}$	M^{60}
$\bar{1}$	$(M^{60})^2 = M^{120}$	$M^{120} \cdot M^{-1} = M^{119}$

The number of squarings plus multiplications is equal to $7 + 2 = 9$ which is 2 less than that of the binary method. The number of squaring operations required by the recoding binary method can be at most 1 more than that of the binary method. The number of subsequent multiplications, on the other hand, can be significantly less. This is simply equal to the number of nonzero digits of the recoded exponent. In the following we describe algorithms for obtaining a sparse signed-digit exponent. These algorithms have been used to obtain efficient multiplication algorithms. It is well-known that the shift-add type of multiplication algorithms perform a shift operation for every bit of the multiplier; an addition is performed if the current bit of the multiplier is equal to 1, otherwise, no operation is performed, and the algorithm proceeds to the next bit. Thus, the number of addition operations can be reduced if we obtain a sparse signed-digit representation of the multiplier. We perform no operation

if the current multiplier bit is equal to 0, an addition if it is equal to 1, and a subtraction if the current bit is equal to $\bar{1}$. These techniques are applicable to exponentiation, where we replace addition by multiplication and subtraction by division, or multiplication with the inverse.

2.10.1 The Booth Algorithm and Modified Schemes

The Booth algorithm [3] scans the bits of the binary number $e = (e_{k-1}e_{k-2}\cdots e_1e_0)$ from right to left, and obtains the digits of the recoded number f using the following truth table:

e_i	e_{i-1}	f_i
0	0	0
0	1	1
1	0	$\bar{1}$
1	1	0

To obtain f_0, we take $e_{-1} = 0$. For example, the recoding of $e = (111001111)$ is obtained as

$$\begin{array}{c} 111001111 \\ 100\bar{1}01000\bar{1} \end{array}$$

which is more sparse than the ordinary exponent. However, the Booth algorithm has a shortcoming: The repeated sequences of (01) are recoded as repated sequences of $(1\bar{1})$. Thus, the resulting number may be much less sparse. The worst case occurs for a number of the form $e = (101010101)$, giving

$$\begin{array}{c} 101010101 \\ 1\bar{1}1\bar{1}1\bar{1}1\bar{1}1\bar{1} \end{array}$$

We are much better off not recoding this exponent. Another problem, which is related to this one, with the Booth algorithm is that when two trails of ones are separated by a zero, the Booth algorithm does not combine them even though they can be combined. For example, the number $e = (11101111)$ is recoded as

$$\begin{array}{c} 11101111 \\ 100\bar{1}1000\bar{1} \end{array}$$

even though a more sparse recoding exists:

$$\begin{array}{c} 100\bar{1}1000\bar{1} \\ 1000\bar{1}000\bar{1} \end{array}$$

since $(\bar{1}1) = -2 + 1 = -1 = (0\bar{1})$. In order to circumvent these shortcomings of the Booth algorithm, several modifications have been proposed [51, 16, 29]. These algorithms scan several bits at a time, and attempt to avoid introducing unnecessary nonzero digits to the recoded number. All of these algorithms which are designed for multiplication are applicable

for exponentiation. Running time analyses of some of these modified Booth algorithms in the context of modular exponentiation have been performed [33, 21]. For example, the modified Booth scheme given in [21] scans the bits of the exponent four bits at a time sharing one bit with the previous and two bits with the next case:

e_{i+1}	e_i	e_{i-1}	e_{i-2}	f_i	e_{i+1}	e_i	e_{i-1}	e_{i-2}	f_i
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	1	0
0	0	1	0	0	1	0	1	0	0
0	0	1	1	1	1	0	1	1	1
0	1	0	0	1	1	1	0	0	$\bar{1}$
0	1	0	1	1	1	1	0	1	$\bar{1}$
0	1	1	0	0	1	1	1	0	0
0	1	1	1	0	1	1	1	1	0

This technique recodes the number in such a way that the isolated 1s stay untouched. Also 0110 is recoded as $10\bar{1}0$ and any trail of 1s of length $i \geq 3$ is recoded as $10\cdots0\bar{1}$. We have shown that the binary method requires $\frac{3}{2}(k-1)$ squarings plus multiplications on the average. The recoding binary method requires significantly fewer multiplications, and the number of squarings is increased by at most 1. In order to count the average number of consequent multiplications, we calculate the probability of the signed-digit value being equal to nonzero, i.e., 1 or $\bar{1}$. For the above recoding scheme, an analysis has been performed in [21]. The recoding binary method using the recoding strategy given in the able requires a total of $\frac{11}{8}(k-1)$ squarings plus multiplications. The average asymptotic savings in the number of squarings plus multiplications is equal to

$$\left(\frac{3}{2}-\frac{11}{8}\right) \div \frac{3}{2} = \frac{1}{12} \approx 8.3\ \% \ .$$

The average number of multiplications plus squarings are tabulated in the following table:

k	binary	recoding
8	11	10
16	23	21
32	47	43
64	95	87
128	191	175
256	383	351
512	767	703
1024	1535	1407
2048	3071	2815

2.10.2 The Canonical Recoding Algorithm

In a signed-digit number with radix 2, three symbols $\{\bar{1}, 0, 1\}$ are allowed for the digit set, in which 1 in bit position i represents $+2^i$ and $\bar{1}$ in bit position i represents -2^i. A

minimal signed-digit vector $f = (f_k f_{k-1} \cdots f_1 f_0)$ that contains no adjacent nonzero digits (i.e. $f_i f_{i-1} = 0$ for $0 < i \leq k$) is called a *canonical signed-digit vector*. If the binary expansion of E is viewed as padded with an initial zero, then it can be proved that there exists a unique canonical signed-digit vector for e [38]. The canonical recoding algorithm [38, 16, 29] computes the signed-digit number

$$f = (f_k f_{k-1} f_{k-2} \cdots f_0)$$

starting from the least significant digit. We set the auxiliary variable $c_0 = 0$ and examine the binary expansion of e two bits at a time. The canonically recoded digit f_i and the next value of the auxiliary binary variable c_{i+1} for $i = 0, 1, 2, \ldots, n$ are computed using the following truth table.

c_i	e_{i+1}	e_i	c_{i+1}	f_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	$\bar{1}$
1	0	0	0	1
1	0	1	1	0
1	1	0	1	$\bar{1}$
1	1	1	1	0

As an example, when $e = 3038$, i.e.,

$$e = (0101111011110) = 2^{11} + 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 \ ,$$

we compute the canonical signed-digit vector f as

$$f = (10\bar{1}0000\bar{1}000\bar{1}0) = 2^{12} - 2^{10} - 2^5 - 2^1 \ .$$

Note that in this example the exponent e contains 9 nonzero bits while its canonically recoded version contains only 4 nonzero digits. Consequently, the binary method requires $11 + 8 = 19$ multiplications to compute M^{3038} when applied to the binary expansion of E, but only $12 + 3 = 15$ multiplications when applied to the canonical signed-digit vector f, provided that $M^{-1} \pmod{n}$ is also supplied. The canonical signed-digit vector f is optimal in the sense that it has the minimum number of nonzero digits among all signed-digit vectors representing the same number. For example, the following signed-digit number for $e = 3038$ produced by the original Booth recoding algorithm contains 5 nonzero digits instead of 4:

$$f = (011000\bar{1}1000\bar{1}0) = 2^{11} + 2^{10} - 2^6 + 2^5 - 2^1 \ .$$

Certain variations of the Booth algorithm also produce recodings which are suboptimal in terms of the number of zero digits of the recoding. For example, the first of the two algorithms given in [33] replaces the occurrences of 01^a0 by $10^{a-1}\bar{1}0$, and consequently recodes

(01111011110) as $(1000\bar{1}1000\bar{1}0)$. Since $(\bar{1}1) = (0\bar{1})$, the optimal recoding is $(10000\bar{1}000\bar{1}0)$. The second algorithm in [33] recodes (01111011110) correctly but is suboptimal on binary numbers in which two trails of 1s are separated by (010). For example (0111101011110) is recoded as $(1000\bar{1}011000\bar{1}0)$, which can be made more sparse by using the identity $(\bar{1}011) = (\bar{1}0\bar{1})$. We note that Reitwiesner's canonical recoding algorithm has none of these shortcomings; the recoding f it produces is provably the optimal signed-digit number [38].

It has been observed that when the exponent is recoded using the canonical bit recoding technique then the average number of multiplications for large k can be reduced to $\frac{4}{3}k + O(1)$ provided that M^{-1} is supplied along with M. This is proved in [11] by using formal languages to model the Markovian nature of the generation of canonically recoded signed-digit numbers from binary numbers and counting the average number of nonzero bits. The average asymptotical savings in the number of squarings plus multiplications is equal to

$$\left(\frac{3}{2} - \frac{4}{3}\right) \div \frac{3}{2} = \frac{1}{9} \approx 11\ \% \ .$$

The average number of squarings plus multiplications are tabulated in the following table:

k	binary	canonical
8	11	11
16	23	22
32	47	43
64	95	86
128	191	170
256	383	342
512	767	683
1024	1535	1366
2048	3071	2731

2.10.3 The Canonical Recoding m-ary Method

The recoding binary methods can be generalized to their respective recoding m-ary counterparts. Once the digits of the exponent are recoded, we scan them more than one bit at a time. In fact, more sophisticated techniques, such as the sliding window technique can also be used to compute $M^e \pmod{n}$ once the recoding of the exponent e is obtained. Since the partitioned exponent values are allowed to be negative numbers as well, during the preprocessing step M^w for certain $w < 0$ may be computed. This is easily accomplished by computing $(M^{-1})^w \pmod{n}$ because $M^{-1} \pmod{n}$ is assumed to be supplied along with M. One hopes that these sophisticated algorithms someday will become useful. The main obstacle in using them in the RSA cryptosystem seems to be that the time required for the computation of $M^{-1} \pmod{n}$ exceeds the time gained by the use of the recoding technique.

An analysis of the canonical recoding m-ary method has been performed in [12]. It is shown that the average number of squarings plus multiplications for the recoding binary

($d = 1$), the recoding quaternary ($d = 2$), and the recoding octal ($d = 3$) methods are equal to

$$T_r(k,1) = \frac{4}{3}k - \frac{4}{3}\ , \quad T_r(k,2) = \frac{4}{3}k - \frac{2}{3}\ , \quad T_r(k,3) = \frac{23}{18}k + \frac{75}{18}\ ,$$

respectively. In comparison, the standard binary, quaternary, and octal methods respectively require

$$T_s(k,1) = \frac{3}{2}k - \frac{3}{2}\ , \quad T_s(k,2) = \frac{11}{8}k - \frac{3}{4}\ , \quad T_s(k,3) = \frac{31}{24}k - \frac{17}{8}$$

multiplications in the average. Furthermore, the average number of squarings plus multiplications for the canonical recoding m-ary method for $m = 2^d$ is equal to

$$T_r(k,d) = k - d + \left(1 - \frac{1}{3\,2^{d-2}}\right)\left(\frac{k}{d} - 1\right) + \frac{1}{3}\left[2^{d+2} + (-1)^{d+1}\right] - 3\ .$$

For large k and fixed d, the behavior of $T_r(k,d)$ and $T_s(k,d)$ of the standard m-ary method is governed by the coefficient of k. In the following table we compare the values $T_r(k,d)/k$ and $T_s(k,d)/k$ for large k.

$d = \log_2 m$	1	2	3	4	5	6	7	8
$T_s(k,d)/k$	1.5000	1.3750	1.2917	1.2344	1.1938	1.1641	1.1417	1.1245
$T_r(k,d)/k$	1.3333	1.3333	1.2778	1.2292	1.1917	1.1632	1.1414	1.1244

We can compute directly from the expressions that for constant d

$$\lim_{k\to\infty} \frac{T_r(k,d)}{T_s(k,d)} = \frac{(d+1)2^d - \frac{4}{3}}{(d+1)2^d - 1} < 1\ .$$

However, it is interesting to note that if we consider the *optimal* values d_s and d_r of d (which depend on k) which minimize the average number of multiplications required by the standard and the recoded m-ary methods, respectively, then

$$\frac{T_r(k,d_r)}{T_s(k,d_s)} > 1$$

for large k. It is shown in [12] that

$$\frac{T_r(k,d_r)}{T_s(k,d_s)} \approx \frac{1 + \frac{1}{d_r}}{1 + \frac{1}{d_s}}$$

for large k, which implies $T_r(k,d_r) > T_s(k,d_s)$. Exact values of d_s and d_r for a given k can be obtained by enumeration. These optimal values of d_s and d_r are given in the following table together with the corresponding values of T_s and T_r for each $k = 128, 256, \ldots, 2048$.

k	d_s	$T_s(k, d_s)$	d_r	$T_r(k, d_r)$
128	4	168	3	168
256	4	326	4	328
512	5	636	4	643
1024	5	1247	5	1255
2048	6	2440	6	2458

In the following figure, we plot the average number of multiplications required by the standard and canonical recoding m-ary methods as a function of d and k.

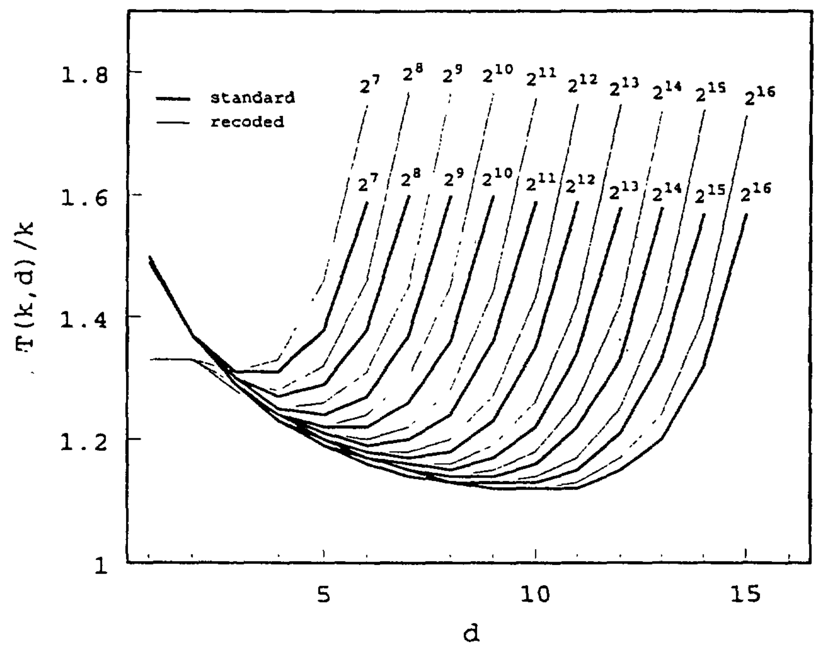


Figure 2.3: The standard versus recoding m-ary methods.

This figure and the previous analysis suggest that the recoding m-ary method may not be as useful as the straightforward m-ary method. A discussion of the usefulness of the recoding exponentiation techniques is found in the following section.

2.10.4 Recoding Methods for Cryptographic Algorithms

The recoding exponentiation methods can perhaps be useful if M^{-1} can be supplied without too much extra cost. Even though the inverse $M^{-1} \pmod{n}$ can easily be computed using the extended Euclidean algorithm, the cost of this computation far exceeds the time gained by the use of the recoding technique in exponentiation. Thus, at this time the recoding techniques do not seem to be particularly applicable to the RSA cryptosystem. In some

contexts, where the plaintext M as well as its inverse M^{-1} modulo n are available for some reason, these algorithms can be quite useful since they offer significant savings in terms of the number multiplications, especially in the binary case. For example, the recoding binary method requires $1.33k$ multiplications while the nonrecoding binary method requires $1.5k$ multiplications. Also, Kaliski [18] has recently shown that if one computes the *Montgomery inverse* instead of the inverse, certain savings can be achieved by making use of the right-shifting binary algorithm. Thus, Kaliski's approach can be utilized for fast computation of the inverse, which opens up new avenues in speeding modular exponentiation computations using the recoding techniques.

On the other hand, the recoding techniques are shown to be useful for computations on elliptic curves over finite fields since in this case the inverse is available at no additional cost [33, 20]. In this context, one computes $e \cdot M$ where e is a large integer and M is a point on the elliptic curve. The multiplication operator is determined by the group law of the elliptic curve. An algorithm for computing M^e is easily converted to an algorithm for computing $e \cdot M$, where we replace multiplication by addition and division (multiplication with the inverse) by subtraction.

Chapter 3

Modular Multiplication

The modular exponentiation algorithms perform modular squaring and multiplication operations at each step of the exponentiation. In order to compute $M^e \pmod{n}$ we need to implement a modular multiplication routine. In this section we will study algorithms for computing

$$R := a \cdot b \pmod{n} ,$$

where a, b, and n are k-bit integers. Since k is often more than 256, we need to build data structures in order to deal with these large numbers. Assuming the word-size of the computer is w (usually $w = 16$ or 32), we break the k-bit number into s words such that $(s-1)w < k \leq sw$. The temporary results may take longer than s words, and thus, they need to be accommodated as well.

3.1 Modular Multiplication

In this report, we consider the following three methods for computing of $R = a \cdot b \pmod{n}$.

- Multiply and then Reduce:

  First Multiply $t := a \cdot b$. Here t is a $2k$-bit or $2s$-word number.

  Then Reduce: $R := t \bmod n$. The result u is a k-bit or s-word number.

  The reduction is accomplished by dividing t by n, however, we are not interested in the quotient; we only need the remainder. The steps of the division algorithm can be somewhat simplified in order to speed up the process.

- Blakley's method:

  The multiplication steps are interleaved with the reduction steps.

- Montgomery's method:

  This algorithm rearranges the residue class modulo n, and uses modulo 2^j arithmetic.

3.2 Standard Multiplication Algorithm

Let a and b be two s-digit (s-word) numbers expressed in radix W as:

$$\begin{aligned} a &= (a_{s-1}a_{s-2}\cdots a_0) = \sum_{j=0}^{s-1} a_i W^i \ , \\ b &= (b_{s-1}b_{s-2}\cdots b_0) = \sum_{j=0}^{s-1} b_i W^i \ , \end{aligned}$$

where the digits of a and b are in the range $[0, W-1]$. In general W can be any positive number. For computer implementations, we often select $W = 2^w$ where w is the word-size of the computer, e.g., $w = 32$. The standard (pencil-and-paper) algorithm for multiplying a and b produces the partial products by multiplying a digit of the multiplier (b) by the entire number a, and then summing these partial products to obtain the final number $2s$-word number t. Let t_{ij} denote the (Carry,Sum) pair produced from the product $a_i \cdot b_j$. For example, when $W = 10$, and $a_i = 7$ and $b_j = 8$, then $t_{ij} = (5,6)$. The t_{ij} pairs can be arranged in a table as

$$\begin{array}{cccccccc} & & & & a_3 & a_2 & a_1 & a_0 \\ \times & & & & b_3 & b_2 & b_1 & b_0 \\ \hline & & & & t_{03} & t_{02} & t_{01} & t_{00} \\ & & & t_{13} & t_{12} & t_{11} & t_{10} & \\ & & t_{23} & t_{22} & t_{21} & t_{20} & & \\ + & t_{33} & t_{32} & t_{31} & t_{30} & & & \\ \hline t_7 & t_6 & t_5 & t_4 & t_3 & t_2 & t_1 & t_0 \end{array}$$

The last row denotes the total sum of the partial products, and represents the product as an $2s$-word number. The standard algorithm for multiplication essentially performs the above digit-by-digit multiplications and additions. In order to save space, a single partial product variable t is being used. The initial value of the partial product is equal to zero; we then take a digit of b and multiply by the entire number a, and add it to the partial product t. The partial product variable t contains the final product $a \cdot b$ at the end of the computation. The standard algorithm for computing the product $a \cdot b$ is given below:

The Standard Multiplication Algorithm
Input: a, b
Output: $t = a \cdot b$

```
0.  Initially t_i := 0 for all i = 0,1,...,2s-1.
1.  for i = 0 to s-1
2.      C := 0
3.      for j = 0 to s-1
4.          (C,S) := t_{i+j} + a_j · b_i + C
5.          t_{i+j} := S
6.      t_{i+s} := C
7.  return (t_{2s-1} t_{2s-2} ··· t_0)
```

In the following, we show the steps of the computation of $a \cdot b = 348 \cdot 857$ using the standard algorithm.

i	j	Step	(C, S)	Partial t
0	0	$t_0 + a_0 b_0 + C$	$(0, *)$	000000
		$0 + 8 \cdot 7 + 0$	$(5, 6)$	000006
	1	$t_1 + a_1 b_0 + C$		
		$0 + 4 \cdot 7 + 5$	$(3, 3)$	000036
	2	$t_2 + a_2 b_0 + C$		
		$0 + 3 \cdot 7 + 3$	$(2, 4)$	000436
				002436
1	0	$t_1 + a_0 b_1 + C$	$(0, *)$	
		$3 + 8 \cdot 5 + 0$	$(4, 3)$	002436
	1	$t_2 + a_1 b_1 + C$		
		$4 + 4 \cdot 5 + 4$	$(2, 8)$	002836
	2	$t_3 + a_2 b_1 + C$		
		$2 + 3 \cdot 5 + 2$	$(1, 9)$	009836
				019836
2	0	$t_2 + a_0 b_2 + C$	$(0, *)$	
		$8 + 8 \cdot 8 + 0$	$(7, 2)$	019236
	1	$t_3 + a_1 b_2 + C$		
		$9 + 4 \cdot 8 + 7$	$(4, 8)$	018236
	2	$t_4 + a_2 b_2 + C$		
		$1 + 3 \cdot 8 + 4$	$(2, 9)$	098236
				298236

In order to implement this algorithm, we need to be able to execute Step 4:

$$(C, S) := t_{i+j} + a_j \cdot b_i + C \ ,$$

where the variables t_{i+j}, a_j, b_i, C, and S each hold a single-word, or a W-bit number. This step is termed as an inner-product operation which is common in many of the arithmetic and number-theoretic calculations. The inner-product operation above requires that we multiply two W-bit numbers and add this product to previous 'carry' which is also a W-bit number and then add this result to the running partial product word t_{i+j}. From these three operations we obtain a $2W$-bit number since the maximum value is

$$2^W - 1 + (2^W - 1)(2^W - 1) + 2^W - 1 = 2^{2W} - 1 \ .$$

Also, since the inner-product step is within the innermost loop, it needs to run as fast as possible. Of course, the best thing is to have a single microprocessor instruction for this computation; unfortunately, none of the currently available microprocessors and signal processors offers such a luxury. A brief inspection of the steps of this algorithm reveals that the total number of inner-product steps is equal to s^2. Since $s = k/w$ and w is a constant on a

given computer, the standard multiplication algorithm requires $O(k^2)$ bit operations in order to multiply two k-bit numbers. This algorithm is asymptotically slower than the Karatsuba algorithm and the FFT-based algorithm which are to be studied next. However, it is simpler to implement and, for small numbers, gives better performance than these asymptotically faster algorithms.

3.3 Karatsubă-Ofman Algorithm

We now describe a recursive algorithm which requires asymptotically fewer than $O(k^2)$ bit operations to multiply two k-bit numbers. The algorithm was introduced by two Russian mathematicians Karatsuba and Ofman in 1962. The details of the Karatsuba-Ofman algorithm can be found in Knuth's book [19]. The following is a brief explanation of the algorithm. First, decompose a and b into two equal-size parts:

$$\begin{aligned} a &:= 2^h a_1 + a_0 \ , \\ b &:= 2^h b_1 + b_0 \ , \end{aligned}$$

i.e., a_1 is higher order h bits of a and a_0 is the lower h bits of a, assuming k is even and $2h = k$. Since we will be worried only about the asymptotics of the algorithm, let us assume that k is a power of 2. The algorithm breaks the multiplication of a and b into multiplication of the parts a_0, a_1, b_0, and b_1. Since

$$\begin{aligned} t &:= a \cdot b \\ &:= (2^h a_1 + a_0)(2^h b_1 + b_0) \\ &:= 2^{2h}(a_1 b_1) + 2^h(a_1 b_0 + a_0 b_1) + a_0 b_0 \\ &:= 2^{2h} t_2 + 2^h t_1 + t_0 \ , \end{aligned}$$

the multiplication of two $2h$-bit numbers seems to require the multiplication of four h-bit numbers. This formulation yields a recursive algorithm which we will call the standard recursive multiplication algorithm (SRMA).

> **function** SRMA(a, b)
> $t_0 :=$ SRMA(a_0, b_0)
> $t_2 :=$ SRMA(a_1, b_1)
> $u_0 :=$ SRMA(a_0, b_1)
> $u_1 :=$ SRMA(a_1, b_0)
> $t_1 := u_0 + u_1$
> **return** $(2^{2h} t_2 + 2^h t_1 + t_0)$

Let $T(k)$ denote the number of bit operations required to multiply two k-bit numbers. Then the standard recursive multiplication algorithm implies that

$$T(k) = 4T(\frac{k}{2}) + \alpha k \ ,$$

where αk denotes the number of bit operations required to compute the addition and shift operations in the above algorithm (α is a constant). Solving this recursion with the initial condition $T(1) = 1$, we find that the standard recursive multiplication algorithm requires $O(k^2)$ bit operations to multiply two k-bit numbers.

The Karatsuba-Ofman algorithm is based on the following observation that, in fact, three half-size multiplications suffice to achieve the same purpose:

$$\begin{aligned} t_0 &:= a_0 \cdot b_0 \ , \\ t_2 &:= a_1 \cdot b_1 \ , \\ t_1 &:= (a_0 + a_1) \cdot (b_0 + b_1) - t_0 - t_2 = a_0 \cdot b_1 + a_1 \cdot b_0 \ . \end{aligned}$$

This yields the Karatsuba-Ofman recursive multiplication algorithm (KORMA) which is illustrated below:

function KORMA(a, b)
$t_0 :=$ KORMA(a_0, b_0)
$t_2 :=$ KORMA(a_1, b_1)
$u_0 :=$ KORMA$(a_1 + a_0, b_1 + b_0)$
$t_1 := u_0 - t_0 - t_2$
return $(2^{2h} t_2 + 2^h t_1 + t_0)$

Let $T(k)$ denote the number of bit operations required to multiply two k-bit numbers using the Karatsuba-Ofman algorithm. Then,

$$T(k) = 2T(\frac{k}{2}) + T(\frac{k}{2} + 1) + \beta k \approx 3T(\frac{k}{2}) + \beta k \ .$$

Similarly, βk represents the contribution of the addition, subtraction, and shift operations required in the recursive Karatsuba-Ofman algorithm. Using the initial condition $T(1) = 1$, we solve this recursion and obtain that the Karatsuba-Ofman algorithm requires

$$O(k^{\log_2 3}) = O(k^{1.58})$$

bit operations in order to multiply two k-bit numbers. Thus, the Karatsuba-Ofman algorithm is asymptotically faster than the standard (recursive as well as nonrecursive) algorithm which requires $O(k^2)$ bit operations. However, due to the recursive nature of the algorithm, there is some overhead involved. For this reason, Karatsuba-Ofman algorithm starts paying off as k gets larger. Current implementations indicate that after about $k = 250$, it starts being faster than the standard nonrecursive multiplication algorithm. Also note that since $a_0 + a_1$ is one bit larger, thus, some implementation difficulties may arise. However, we also have the option of stopping at any point during the recursion. For example, we may apply one level of recursion and then compute the required three multiplications using the standard nonrecursive multiplication algorithm.

3.4 FFT-based Multiplication Algorithm

The fastest multiplication algorithms use the fast Fourier transform. Although the fast Fourier transform was originally developed for convolution of sequences, which amounts to multiplication of polynomials, it can also be used for multiplication of long integers. In the standard algorithm, the integers are represented by the familiar positional notation. This is equivalent to polynomials to be evaluated at the radix; for example, $348 = 3x^2 + 4x + 8$ at $x = 10$. Similarly, $857 = 8x^2 + 5x + 7$ at $x = 10$. In order to multiply 348 by 857, we can first multiply the polynomials

$$(3x^2 + 4x + 8)(8x^2 + 5x + 7) = 24x^4 + 47x^3 + 105x^2 + 68x + 56 ,$$

then evaluate the resulting polynomial

$$24(10)^4 + 47(10)^3 + 105(10)^2 + 68(10) + 56 = 298236$$

at 10 to obtain the product $348 \cdot 857 = 298236$. Therefore, if we can multiply polynomials quickly, then we can multiply large integers quickly. In order to multiply two polynomials, we utilize the discrete Fourier transform. This is achieved by evaluating these polynomials at the roots of unity, then multiplying these values pointwise, and finally interpolating these values to obtain the coefficients of the product polynomial. The fast Fourier transform algorithm allows us to evaluate a given polynomial of degree $s-1$ at the s roots of unity using $O(s \log s)$ arithmetic operations. Similarly, the interpolation step is performed in $O(s \log s)$ time.

A polynomial is determined by its coefficients. Moreover, there exists a unique polynomial of degree $s - 1$ which 'visits' s points on the plane provided that the axes of these points are distinct. These s pairs of points can also be used to uniquely represent the polynomial of degree $s - 1$. Let $A(x)$ be a polynomial of degree $l - 1$, i.e.,

$$A(x) = \sum_{i=0}^{l-1} A_i x^i .$$

Also, let ω be the primitive lth root of unity. Then the fast Fourier transform algorithm can be used to evaluate this polynomial at $\{1, \omega, \omega^2, \ldots, \omega^{l-1}\}$ using $O(l \log l)$ arithmetic operations [31]. In other words, the fast Fourier transform algorithm computes the matrix vector product

$$\begin{bmatrix} A(1) \\ A(\omega) \\ \vdots \\ A(\omega^{l-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{l-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{l-1} & \cdots & \omega^{(l-1)(l-1)} \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{l-1} \end{bmatrix} ,$$

in order to obtain the polynomial values $A(\omega^i)$ for $i = 0, 1, \ldots, l-1$. These polynomial values also uniquely define the polynomial $A(x)$. Given these polynomial values, the coefficients A_i

for $i = 0, 1, \ldots, l-1$ can be obtained by the use of the 'inverse' Fourier transform:

$$\begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{l-1} \end{bmatrix} = l^{-1} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \cdots & \omega^{-(l-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(l-1)} & \cdots & \omega^{-(l-1)(l-1)} \end{bmatrix} \begin{bmatrix} A(1) \\ A(\omega) \\ \vdots \\ A(\omega^{l-1}) \end{bmatrix},$$

where l^{-1} and ω^{-1} are the inverses of l and ω, respectively. The polynomial multiplication algorithm utilizes these subroutines. Let the polynomials $a(x)$ and $b(x)$

$$a(x) = \sum_{i=0}^{s-1} a_i x^i \ , \quad b(x) = \sum_{i=0}^{s-1} b_i x^i$$

denote the multiprecision numbers $a = (a_{s-1}a_{s-2}\cdots a_0)$ $b = (b_{s-1}b_{s-2}\cdots b_0)$ represented in radix W where a_i and b_i are the 'digits' with the property $0 \leq a_i, b_i \leq W-1$. Let the integer $l = 2s$ be a power of 2. Given the primitive lth root of unity ω, the following algorithm computes the product $t = (t_{l-1}t_{l-2}\cdots t_0)$.

FFT-based Integer Multiplication Algorithm

Step 1. Evaluate $a(\omega^i)$ and $b(\omega^i)$ for $i = 0, 1, \ldots, l-1$ by calling the fast Fourier transform procedure.

Step 2. Multiply pointwise to obtain

$$\{a(1)b(1), a(\omega)b(\omega), \ldots, a(\omega^{l-1})b(\omega^{l-1})\} \ .$$

Step 3. Interpolate $t(x) = \sum_{i=0}^{l-1} t_i x^i$ by evaluating

$$l^{-1} \sum_{i=0}^{l-1} a(\omega^i) b(\omega^i) x^i$$

on $\{1, \omega^{-1}, \ldots, \omega^{-(l-1)}\}$ using the fast Fourier transform procedure.

Step 4. Return the coefficients $(t_{l-1}, t_{l-2}, \ldots, t_0)$.

The above fast integer multiplication algorithm works over an arbitrary field in which l^{-1} and a primitive lth root of unity exist. Here, the most important question is which field to use. The fast Fourier transform was originally developed for the field of complex numbers in which the familiar lth root of unity $e^{2\pi j/l}$ makes this field the natural choice (here, $j = \sqrt{-1}$). However, there are computational difficulties in the use of complex numbers. Since computers can only perform finite precision arithmetic, we may not be able perform arithmetic with quantities such as $e^{2\pi j/l}$ because these numbers may be irrational.

In 1971, Pollard [36] showed that any field can be used provided that l^{-1} and a primitive lth root of unity are available. We are especially interested in finite fields, since our computers perform finite precision arithmetic. The field of choice is the Galois field of p elements where p is a prime and l divides $p-1$. This is due to the theorem which states that if p be prime and l divides $p-1$, then l^{-1} is in $GF(p)$ and $GF(p)$ has a primitive lth root of unity. Fortunately, such primes p are not hard to find. Primes of the form $2^r s+1$, where s is odd, have been listed in books, e.g, in [39]. Their primitive roots are readily located by successively testing. There exist an abundance of primes in the arithmetic progression $2^r s + 1$, and primitive roots make up more than 3 out of every π^2 elements in the range from 2 to $p-1$ [31, 7]. For example, there are approximately 180 primes $p = 2^r s + 1 < 2^{31}$ with $r \geq 20$. Any such prime can be used to compute the fast Fourier transform of size 2^{20} [31]. Their primitive roots may also be found in a reasonable amount of time. The following list are the 10 largest primes of the form $p = 2^r s + 1 \leq 2^{31} - 1$ with $r > 20$ and their least primitive roots α.

p	r	α
2130706433	24	3
2114977793	20	3
2113929217	25	5
2099249153	21	3
2095054849	21	11
2088763393	23	5
2077229057	20	3
2070937601	20	6
2047868929	20	13
2035286017	20	10

The primite lth root of unity can easily be computed from α using $\alpha^{(p-1)/l}$. Thus, mod p FFT computations are viable. There are many Fourier primes, i.e., primes p for which FFTs in modulo p arithmetic exist. Moreover, there exists a reasonably efficient algorithm for determining such primes along with their primitive elements [31]. From these primitive elements, the required primitive roots of unity can be efficiently computed. This method for multiplication of long integers using the fast Fourier transform over finite fields was discovered by Schönhage and Strassen [45]. It is described in detail by Knuth [19]. A careful analysis of the algorithm shows that the product of two k-bit numbers can be performed using $O(k \log k \log\log k)$ bit operations. However, the constant in front of the order function is high. The break-even point is much higher than that of Karatsuba-Ofman algorithm. It starts paying off for numbers with several thousand bits. Thus, they are not very suitable for performing RSA operations.

3.5 Squaring is Easier

Squaring is an easier operation than multiplication since half of the single-precision multiplications can be skipped. This is due to the fact that $t_{ij} = a_i \cdot a_j = t_{ji}$.

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
\times & & & & a_3 & a_2 & a_1 & a_0 \\
\hline
 & & & & t_{03} & t_{02} & t_{01} & t_{00} \\
 & & & t_{13} & t_{12} & t_{11} & t_{01} & \\
 & & t_{23} & t_{22} & t_{12} & t_{02} & & \\
+ & t_{33} & t_{23} & t_{13} & t_{03} & & & \\
\hline
 & & & & 2t_{03} & 2t_{02} & 2t_{01} & t_{00} \\
 & & & 2t_{13} & 2t_{12} & t_{11} & & \\
 & & 2t_{23} & t_{22} & & & & \\
+ & t_{33} & & & & & & \\
\hline
t_7 & t_6 & t_5 & t_4 & t_3 & t_2 & t_1 & t_0
\end{array}
$$

Thus, we can modify the standard multiplication procedure to take advantage of this property of the squaring operation.

The Standard Squaring Algorithm
Input: a
Output: $t = a \cdot a$

0. Initially $t_i := 0$ for all $i = 0, 1, \ldots, 2s-1$.
1. **for** $i = 0$ **to** $s - 1$
2. $\quad (C, S) := t_{i+i} + a_i \cdot a_i$
3. $\quad$ **for** $j = i + 1$ **to** $s - 1$
4. $\quad\quad (C, S) := t_{i+j} + 2 \cdot a_j \cdot a_i + C$
5. $\quad\quad t_{i+j} := S$
6. $\quad t_{i+s} := C$
7. **return** $(t_{2s-1} t_{2s-2} \cdots t_0)$

However, we warn the reader that the carry-sum pair produced by operation

$$(C, S) := t_{i+j} + 2 \cdot a_j \cdot a_i + C$$

in Step 4 may be 1 bit longer than a single-precision number which requires w bits. Since

$$(2^w - 1) + 2(2^w - 1)(2^w - 1) + (2^w - 1) = 2^{2w+1} - 2^{w+1}$$

and

$$2^{2w} - 1 < 2^{2w+1} - 2^{w+1} < 2^{2w+1} - 1 \ ,$$

the carry-sum pair requires $2w + 1$ bits instead of $2w$ bits for its representation. Thus, we need to accommodate this 'extra' bit during the execution of the operations in Steps 4, 5, and 6. The resolution of this carry may depend on the way the carry bits are handled by the particular processor's architecture. This issue, being rather implementation-dependent, will not be discussed here.

3.6 Computation of the Remainder

The multiply-and-reduce modular multiplication algorithm first computes the product $a \cdot b$ (or, $a \cdot a$) using one of the multiplication algorithms given above. The multiplication step is then followed by a division algorithm in order to compute the remainder. However, as we have noted in Section 3.1, we are not interested in the quotient; we only need the remainder. Therefore, the steps of the division algorithm can somewhat be simplified in order to speed up the process. The reduction step can be achieved by making one of the well-known sequential division algorithms. In the following sections, we describe the restoring and the nonrestoring division algorithms for computing the remainder of t when divided by n.

Division is the most complex of the four basic arithmetic operations. First of all, it has two results: the quotient and the remainder. Given a dividend t and a divisor n, a quotient Q and a remainder R have to be calculated in order to satisfy

$$t = Q \cdot n + R \text{ with } \quad R < n \ .$$

If t and n are positive, then the quotient Q and the remainder R will be positive. The sequential division algorithm successively shifts and subtracts n from t until a remainder R with the property $0 \leq R < n$ is found. However, after a subtraction we may obtain a negative remainder. The restoring and nonrestoring algorithms take different actions when a negative remainder is obtained.

3.6.1 Restoring Division Algorithm

Let R_i be the remainder obtained during the ith step of the division algorithm. Since we are not interested in the quotient, we ignore the generation of the bits of the quotient in the following algorithm. The procedure given below first left-aligns the operands t and n. Since t is $2k$-bit number and n is a k-bit number, the left alignment implies that n is shifted k bits to the left, i.e., we start with $2^k n$. Furthermore, the initial value of R is taken to be t, i.e., $R_0 = t$. We then subtract the shifted n from t to obtain R_1; if R_1 is positive or zero, we continue to the next step. If it is negative the remainder is restored to its previous value.

The Restoring Division Algorithm
Input: t, n
Output: $R = a \bmod n$
1. $R_0 := t$
2. $n := 2^k n$
3. **for** $i = 1$ **to** k
4. $\quad R_i := R_{i-1} - n$
5. $\quad$ **if** $R_i < 0$ **then** $R_i := R_{i-1}$
6. $\quad n := n/2$
7. **return** R_k

In Step 5 of the algorithm, we check the sign of the remainder; if it is negative, the previous remainder is taken to be the new remainder, i.e., a restore operation is performed. If the remainder R_i is positive, it remains as the new remainder, i.e., we do not restore. The restoring division algorithm performs k subtractions in order to reduce the $2k$-bit number t modulo the k-bit number n. Thus, it takes much longer than the standard multiplication algorithm which requires $s = k/w$ inner-product steps, where w is the word-size of the computer.

In the following, we give an example of the restoring division algorithm for computing $3019 \bmod 53$, where $3019 = (101111001011)_2$ and $53 = (110101)_2$. The result is $51 = (110011)_2$.

R_0		101111	001011	t
n		110101		subtract
	$-$	000110		negative remainder
R_1		101111	001011	restore
$n/2$		11010	1	shift and subtract
	$+$	10100	1	positive remainder
R_2		10100	101011	not restore
$n/2$		1101	01	shift and subtract
	$+$	0111	01	positive remainder
R_3		0111	011011	not restore
$n/2$		110	101	shift and subtract
	$+$	000	110	positive remainder
R_4		000	110011	not restore
$n/2$		11	0101	shift
$n/2$		1	10101	shift
$n/2$			110101	shift and subtract
	$+$		000010	negative remainder
R_5			110011	restore
R			110011	final remainder

Also, before subtracting, we may check if the most significant bit of the remainder is 1. In this case, we perform a subtraction. If it is zero, there is no need to subtract since $n > R_i$. We shift n until it is aligned with a nonzero most significant bit of R_i. This way we are able to skip several subtract/restore cycles. In the average, $k/2$ subtractions are performed.

3.6.2 Nonrestoring Division Algorithm

The nonrestoring division algorithm allows a negative remainder. In order to correct the remainder, a subtraction or an addition is performed during the next cycle, depending on the whether the sign of the remainder is positive or negative, respectively. This is based on the following observation: Suppose $R_i = R_{i-1} - n < 0$, then the restoring algorithm assigns

$R_i := R_{i-1}$ and performs a subtraction with the shifted n, obtaining

$$R_{i+1} = R_i - n/2 = R_{i-1} - n/2 \ .$$

However, if $R_i = R_{i-1} - n < 0$, then one can instead let R_i remain negative and add the shifted n in the following cycle. Thus, one obtains

$$R_{i+1} = R_i + n/2 = (R_{i-1} - n) + n/2 = R_{i-1} - n/2 \ ,$$

which would be the same value. The steps of the nonrestoring algorithm, which implements this observation, are given below:

The Nonrestoring Division Algorithm
Input: t, n
Output: $R = t \bmod n$

1. $R_0 := t$
2. $n := 2^k n$
3. **for** $i = 1$ **to** k
4.   **if** $R_{i-1} > 0$ **then** $R_i := R_{i-1} - n$
5.   **else** $R_i := R_{i-1} + n$
6. $n := n/2$
7. **if** $R_k < 0$ **then** $R := R + n$
8. **return** R_k

Note that the nonrestoring division algorithm requires a final restoration cycle in which a negative remainder is corrected by adding the last value of n back to it. In the following we compute $51 = 3019 \bmod 53$ using the nonrestoring division algorithm. Since the remainder is allowed to stay negative, we use 2's complement coding to represent such numbers.

R_0	0101111	001011	t
n	0110101		subtract
R_1	1111010		negative remainder
$n/2$	011010	1	add
R_2	010100	1	positive remainder
$n/2$	01101	01	subtract
R_3	00111	01	positive remainder
$n/2$	0110	101	subtract
R_4	0000	110	positive remainder
$n/2$	011	0101	
$n/2$	01	10101	
$n/2$	0	110101	subtract
R_5	1	111110	negative remainder
n	0	110101	add (final restore)
R	0	110011	Final remainder

3.7 Blakley's Method

Blakley's method [2, 47] directly computes $a \cdot b \bmod n$ by interleaving the shift-add steps of the multiplication and the shift-subtract steps of the division. Since the division algorithm proceeds bit-by-bit, the steps of the multiplication algorithm must also follow this process. This implies that we use a bit-by-bit multiplication algorithm rather than a word-by-word multiplication algorithm which would be much quicker. However, the bit-by-bit multiplication algorithms can be made run faster by employing bit-recoding techniques. Furthermore, the m-ary segmentation of the operands and canonical recoding of the multiplier allows much faster implementations [27]. In the following we describe the steps of Blakley's algorithm. Let a_i and b_i represent the bits of the k-bit numbers a and b, respectively. Then, the product t which is a $2k$-bit number can be written as

$$t = a \cdot b = \left(\sum_{i=0}^{k-1} a_i 2^i \right) \cdot b = \sum_{i=0}^{k-1} (a_i \cdot b) 2^i \ .$$

Blakley's algorithm is based on the above formulation of the product t, however, at each step, we perform a reduction in order to make sure that the remainder is less than n. The reduction step may involve several subtractions.

The Blakley Algorithm
Input: a, b, n
Output: $R = a \cdot b \bmod n$
1. $R := 0$
2. **for** $i = 0$ **to** $k-1$
3. $\quad R := 2R + a_{k-1-i} \cdot b$
4. $\quad R := R \bmod n$
5. **return** R

At Step 3, the partial remainder is shifted one bit to the right and the product $a_{k-1-i}b$ is added to the result. This is a step of the right-to-left multiplication algorithm. Let us assume that $0 \leq a, b, R \leq n-1$. Then the new R will be in the range $0 \leq R \leq 3n-3$ since Step 3 of the algorithm implies

$$R := 2R + a_j \cdot b \leq 2(n-1) + (n-1) = 3n - 3 \ ,$$

i.e., at most 2 subtractions will be needed to bring the new R to the range $[0, n-1]$. Thus, Step 4 of the algorithm can be expanded as:

4.1 If $R \geq n$ then $R := R - n$
4.2 If $R \geq n$ then $R := R - n$

This algorithm computes the remainder R in k steps, where at each step one left shift, one addition, and at most two subtractions are performed; the operands involved in these computations are k-bit binary numbers.

3.8 Montgomery's Method

In 1985, P. L. Montgomery introduced an efficient algorithm [32] for computing $R = a \cdot b \bmod n$ where a, b, and n are k-bit binary numbers. The algorithm is particularly suitable for implementation on general-purpose computers (signal processors or microprocessors) which are capable of performing fast arithmetic modulo a power of 2. The Montgomery reduction algorithm computes the resulting k-bit number R without performing a division by the modulus n. Via an ingenious representation of the residue class modulo n, this algorithm replaces division by n operation with division by a power of 2. This operation is easily accomplished on a computer since the numbers are represented in binary form. Assuming the modulus n is a k-bit number, i.e., $2^{k-1} \leq n < 2^k$, let r be 2^k. The Montgomery reduction algorithm requires that r and n be relatively prime, i.e., $\gcd(r, n) = \gcd(2^k, n) = 1$. This requirement is satisfied if n is odd. In the following we summarize the basic idea behind the Montgomery reduction algorithm.

Given an integer $a < n$, we define its n-residue with respect to r as

$$\bar{a} = a \cdot r \bmod n \ .$$

It is straightforward to show that the set

$$\{\ i \cdot r \bmod n \mid 0 \leq i \leq n-1\ \}$$

is a complete residue system, i.e., it contains all numbers between 0 and $n-1$. Thus, there is a one-to-one correspondence between the numbers in the range 0 and $n-1$ and the numbers in the above set. The Montgomery reduction algorithm exploits this property by introducing a much faster multiplication routine which computes the n-residue of the product of the two integers whose n-residues are given. Given two n-residues $\bar{a}$ and $\bar{b}$, the *Montgomery product* is defined as the n-residue

$$\bar{R} = \bar{a} \cdot \bar{b} \cdot r^{-1} \bmod n$$

where r^{-1} is the inverse of r modulo n, i.e., it is the number with the property

$$r^{-1} \cdot r = 1 \bmod n \ .$$

The resulting number $\bar{R}$ is indeed the n-residue of the product

$$R = a \cdot b \bmod n$$

since

$$\begin{aligned}
\bar{R} &= \bar{a} \cdot \bar{b} \cdot r^{-1} \bmod n \\
&= a \cdot r \cdot b \cdot r \cdot r^{-1} \bmod n \\
&= a \cdot b \cdot r \bmod n \ .
\end{aligned}$$

In order to describe the Montgomery reduction algorithm, we need an additional quantity, n', which is the integer with the property

$$r \cdot r^{-1} - n \cdot n' = 1 \ .$$

The integers r^{-1} and n' can both be computed by the extended Euclidean algorithm [19]. The Montgomery product algorithm, which computes

$$u = \bar{a} \cdot \bar{b} \cdot r^{-1} \pmod{n}$$

given $\bar{a}$ and $\bar{b}$, is given below:

function MonPro($\bar{a}, \bar{b}$)
Step 1. $t := \bar{a} \cdot \bar{b}$
Step 2. $m := t \cdot n' \bmod r$
Step 3. $u := (t + m \cdot n)/r$
Step 4. **if** $u \geq n$ **then return** $u - n$
else return u

The most important feature of the Montgomery product algorithm is that the operations involved are multiplications modulo r and divisions by r, both of which are intrinsically fast operations since r is a power 2. The MonPro algorithm can be used to compute the product of a and b modulo n, provided that n is odd.

function ModMul(a, b, n) { n is an odd number }
Step 1. Compute n' using the extended Euclidean algorithm.
Step 2. $\bar{a} := a \cdot r \bmod n$
Step 3. $\bar{b} := b \cdot r \bmod n$
Step 4. $\bar{x} :=$ MonPro($\bar{a}, \bar{b}$)
Step 5. $x :=$ MonPro($\bar{x}, 1$)
Step 6. **return** x

A better algorithm can be given by observing the property

$$\text{MonPro}(\bar{a}, b) = (a \cdot r) \cdot b \cdot r^{-1} = a \cdot b \pmod{n} \ ,$$

which modifies the above algorithm as

function ModMul(a, b, n) { n is an odd number }
Step 1. Compute n' using the extended Euclidean algorithm.
Step 2. $\bar{a} := a \cdot r \bmod n$
Step 3. $x :=$ MonPro($\bar{a}, b$)
Step 4. **return** x

However, the preprocessing operations, especially the computation of n', are rather time-consuming. Thus, it is not a good idea to use the Montgomery product computation algorithm when a single modular multiplication is to be performed..

3.8.1 Montgomery Exponentiation

The Montgomery product algorithm is more suitable when several modular multiplications with respect to the same modulus are needed. Such is the case when one needs to compute a modular exponentiation, i.e., the computation of M^e mod n. Using one of the addition chain algorithms given in Chapter 2, we replace the exponentiation operation by a series of square and multiplication operations modulo n. This is where the Montgomery product operation finds its best use. In the following we summarize the modular exponentiation operation which makes use of the Montgomery product function MonPro. The exponentiation algorithm uses the binary method.

function ModExp(M, e, n) { n is an odd number }
Step 1. Compute n' using the extended Euclidean algorithm.
Step 2. $\bar{M} := M \cdot r \bmod n$
Step 3. $\bar{x} := 1 \cdot r \bmod n$
Step 4. **for** $i = k - 1$ **down to** 0 **do**
Step 5. $\quad\bar{x} := \text{MonPro}(\bar{x}, \bar{x})$
Step 6. $\quad$**if** $e_i = 1$ **then** $\bar{x} := \text{MonPro}(\bar{M}, \bar{x})$
Step 7. $x := \text{MonPro}(\bar{x}, 1)$
Step 8. **return** x

Thus, we start with the ordinary residue M and obtain its n-residue $\bar{M}$ using a division-like operation, which can be achieved, for example, by a series of shift and subtract operations. Additionally, Steps 2 and 3 require divisions. However, once the preprocessing has been completed, the inner-loop of the binary exponentiation method uses the Montgomery product operations which performs only multiplications modulo 2^k and divisions by 2^k. When the binary method finishes, we obtain the n-residue $\bar{x}$ of the quantity $x = M^e \bmod n$. The ordinary residue number is obtained from the n-residue by executing the MonPro function with arguments $\bar{x}$ and 1. This is easily shown to be correct since

$$\bar{x} = x \cdot r \bmod n$$

immediately implies that

$$x = \bar{x} \cdot r^{-1} \bmod n \ = \ \bar{x} \cdot 1 \cdot r^{-1} \bmod n \ := \ \text{MonPro}(\bar{x}, 1) \ .$$

The resulting algorithm is quite fast as was demonstrated by many researchers and engineers who have implemented it, for example, see [10, 30]. However, this algorithm can be refined and made more efficient, particularly when the numbers involved are multi-precision integers. For example, Dussé and Kaliski [10] gave improved algorithms, including a simple and efficient method for computing n'. We wil describe these methods in Section 4.2.

3.8.2 An Example of Exponentiation

Here we show how to compute $x = 7^{10} \bmod 13$ using the Montgomery exponentiation algorithm.

- Since $n = 13$, we take $r = 2^4 = 16 > n$.
- Computation of n':

  Using the extended Euclidean algorithm, we determine that $16 \cdot 9 - 13 \cdot 11 = 1$, thus, $r^{-1} = 9$ and $n' = 11$.
- Computation of $\bar{M}$:

  Since $M = 7$, we have $\bar{M} := M \cdot r \pmod{n} = 7 \cdot 16 \pmod{13} = 8$.
- Computation of $\bar{x}$ for $x = 1$:

  We have $\bar{x} := x \cdot r \pmod{n} = 1 \cdot 16 \pmod{13} = 3$.
- Steps 5 and 6 of the ModExp routine:

e_i	Step 5	Step 6
1	MonPro(3,3) = 3	MonPro(8,3) = 8
0	MonPro(8,8) = 4	
1	MonPro(4,4) = 1	MonPro(8,1) = 7
0	MonPro(7,7) = 12	

  ∘ Computation of MonPro(3,3) = 3:
  $t := 3 \cdot 3 = 9$
  $m := 9 \cdot 11 \pmod{16} = 3$
  $u := (9 + 3 \cdot 13)/16 = 48/16 = 3$

  ∘ Computation of MonPro(8,3) = 8:
  $t := 8 \cdot 3 = 24$
  $m := 24 \cdot 11 \pmod{16} = 8$
  $u := (24 + 8 \cdot 13)/16 = 128/16 = 8$

  ∘ Computation of MonPro(8,8) = 4:
  $t := 8 \cdot 8 = 64$
  $m := 64 \cdot 11 \pmod{16} = 0$
  $u := (64 + 0 \cdot 13)/16 = 64/16 = 4$

  ∘ Computation of MonPro(4,4) = 1:
  $t := 4 \cdot 4 = 16$
  $m := 16 \cdot 11 \pmod{16} = 0$
  $u := (16 + 0 \cdot 13)/16 = 16/16 = 1$

  ∘ Computation of MonPro(8,1) = 7:
  $t := 8 \cdot 1 = 8$
  $m := 8 \cdot 11 \pmod{16} = 8$
  $u := (8 + 8 \cdot 13)/16 = 112/16 = 7$

  ∘ Computation of MonPro(7,7) = 12:
  $t := 7 \cdot 7 = 49$
  $m := 49 \cdot 11 \pmod{16} = 11$
  $u := (49 + 11 \cdot 13)/16 = 192/16 = 12$

- Step 7 of the ModExp routine: $x = \text{MonPro}(12,1) = 4$
  $t := 12 \cdot 1 = 12$
  $m := 12 \cdot 11 \pmod{16} = 4$
  $u := (12 + 4 \cdot 13)/16 = 64/16 = 4$

Thus, we obtain $x = 4$ as the result of the operation $7^{10} \bmod 13$.

3.8.3 The Case of Even Modulus

Since the existence of r^{-1} and n' requires that n and r be relatively prime, we cannot use the Montgomery product algorithm when this rule is not satisfied. We take $r = 2^k$ since arithmetic operations are based on binary arithmetic modulo 2^w where w is the word-size of the computer. In case of single-precision integers, we take $k = w$. However, when the numbers are large, we choose k to be an integer multiple of w. Since $r = 2^k$, the Montgomery modular exponentiation algorithm requires that

$$\gcd(r, n) = \gcd(2^k, n) = 1$$

which is satisfied if and only if n is odd. We now describe a simple technique [22] which can be used whenever one needs to compute modular exponentiation with respect to an even modulus. Let n be factored such that

$$n = q \cdot 2^j$$

where q is an odd integer. This can easily be accomplished by shifting the even number n to the right until its least-significant bit becomes one. Then, by the application of the Chinese remainder theorem, the computation of

$$x = a^e \bmod n$$

is broken into two independent parts such that

$$\begin{aligned} x_1 &= a^e \bmod q \ , \\ x_2 &= a^e \bmod 2^j \ . \end{aligned}$$

The final result x has the property

$$\begin{aligned} x &= x_1 \bmod q \ , \\ x &= x_2 \bmod 2^j \ , \end{aligned}$$

and can be found using one of the Chinese remainder algorithms: The single-radix conversion algorithm or the mixed-radix conversion algorithm [49, 19, 31]. The computation of x_1 can be performed using the ModExp algorithm since q is odd. Meanwhile the computation of x_2 can be performed even more easily since it involves arithmetic modulo 2^j. There is however some overhead involved due to the introduction of the Chinese remainder theorem. According to the mixed-radix conversion algorithm, the number whose residues are x_1 and x_2 modulo q and 2^j, respectively, is equal to

$$x = x_1 + q \cdot y$$

where

$$y = (x_2 - x_1) \cdot q^{-1} \bmod 2^j \ .$$

The inverse $q^{-1} \bmod 2^j$ exists since q is odd. It can be computed using the simple algorithm given in Section 4.2. We thus have the following algorithm:

function EvenModExp(a, e, n) { n is an even number }
1. Shift n to the right obtain the factorization $n = q \cdot 2^j$.
2. Compute $x_1 := a^e \bmod q$ using ModExp routine above.
3. Compute $x_2 := a^e \bmod 2^j$ using the binary method and modulo 2^j arithmetic.
4. Compute $q^{-1} \bmod 2^j$ and $y := (x_2 - x_1) \cdot q^{-1} \bmod 2^j$.
5. Compute $x := x_1 + q \cdot y$ and **return** x.

3.8.4 An Example of Even Modulus Case

The computation of $a^e \bmod n$ for $a = 375$, $e = 249$, and $n = 388$ is illustrated below.

Step 1. $n = 388 = (110000100)_2 = (11000001)_2 \times 2^2 = 97 \times 2^2$. Thus, $q = 97$ and $j = 2$.

Step 2. Compute $x_1 = a^e \bmod q$ by calling ModExp with parameters $a = 375$, $e = 249$, and $q = 97$. We must remark, however, that we can reduce a and e modulo q and $\phi(q)$, respectively. The latter is possible if we know the factorization of q. Such knowledge is not necessary but would further decrease the computation time of the ModExp routine. Assuming we do not know the factorization of q, we only reduce a to obtain

$$a \bmod q \quad = 375 \bmod 97 \quad = 84$$

and call the ModExp routine with parameters $(84, 249, 97)$. Since q is odd, the ModExp routine successfully computes the result as $x_1 = 78$.

Step 3. Compute $x_2 = a^e \bmod 2^j$ by calling an exponentiation routine based on the binary method and modulo 2^j arithmetic. Before calling such routine we should reduce the parameters as

$$\begin{array}{lll} a \bmod 2^j & = 375 \bmod 4 & = 3 \\ e \bmod \phi(2^j) & = 249 \bmod 2 & = 1 \end{array}$$

In this case, we are able to reduce the exponent since we know that $\phi(2^j) = 2^{j-1}$. Thus, we call the exponentiation routine with the parameters $(3, 1, 4)$. The routine computes the result as $x_2 = 3$.

Step 4. Using the extended Euclidean algorithm, compute

$$q^{-1} \bmod 2^j = 97^{-1} \bmod 4 = 1 \ .$$

Now compute

$$\begin{aligned} y &= (x_2 - x_1) \cdot q^{-1} \bmod 2^j \\ &= (3 - 78) \cdot 1 \bmod 4 \\ &= 1 \ . \end{aligned}$$

Step 5. Compute and return the final result

$$x = x_1 + q \cdot y = 78 + 97 \cdot 1 = 175 \ .$$

Chapter 4

Further Improvements and Performance Analysis

4.1 Fast Decryption using the CRT

The RSA decryption and signing operation, i.e., given C, the computation of

$$M := C^d \pmod{n} ,$$

can be performed faster using the Chinese remainder theorem (CRT) since the user knows the factors of the modulus: $n = p \cdot q$. This method was proposed by Quisquater and Couvreur [37], and is based on the Chincse remainder theorem, another number theory gem, like the binary method, coming to us from antiquity. Let p_i for $i = 1, 2, \ldots, k$ be pairwise relatively prime integers, i.e.,

$$\gcd(p_i, p_j) = 1 \text{ for } i \neq j .$$

Given $u_i \in [0, p_i - 1]$ for $i = 1, 2, \ldots, k$, the Chinese remainder theorem states that there exists a unique integer u in the range $[0, P - 1]$ where $P = p_1 p_2 \cdots p_k$ such that

$$u = u_i \pmod{p_i} .$$

The Chinese remainder theorem tells us that the computation of

$$M := C^d \pmod{p \cdot q} ,$$

can be broken into two parts as

$$\begin{aligned} M_1 &:= C^d \pmod{p} , \\ M_2 &:= C^d \pmod{q} , \end{aligned}$$

after which the final value of M is computed (lifted) by the application of a Chinese remainder algorithm. There are two algorithms for this computation; The single-radix conversion

(SRC) algorithm and the mixed-radix conversion (MRC) algorithm. Here, we briefly describe these algorithms, details of which can be found in [14, 49, 19, 31]. Going back to the general example, we observe that the SRC or the MRC algorithm computes u given $u_1, u_2, \ldots, u_k$ and $p_1, p_2, \ldots, p_k$. The SRC algorithm computes u using the summation

$$u = \sum_{i=1}^{k} u_i c_i P_i \pmod{P} ,$$

where

$$P_i = p_1 p_2 \cdots p_{i-1} p_{i+1} \cdots p_k = \frac{P}{p_i} ,$$

and c_i is the multiplicative inverse of P_i modulo p_i, i.e.,

$$c_i P_i = 1 \pmod{p_i} .$$

Thus, applying the SRC algorithm to the RSA decryption, we first compute

$$\begin{aligned} M_1 &:= C^d \pmod{p} , \\ M_2 &:= C^d \pmod{q} , \end{aligned}$$

However, applying Fermat's theorem to the exponents, we only need to compute

$$\begin{aligned} M_1 &:= C^{d_1} \pmod{p} , \\ M_2 &:= C^{d_2} \pmod{q} , \end{aligned}$$

where

$$\begin{aligned} d_1 &:= d \bmod (p-1) , \\ d_2 &:= d \bmod (q-1) . \end{aligned}$$

This provides some savings since $d_1, d_2 < d$; in fact, the sizes of d_1 and d_2 are about half of the size of d. Proceeding with the SRC algorithm, we compute M using the sum

$$M = M_1 c_1 \frac{pq}{p} + M_2 c_2 \frac{pq}{q} \pmod{n} = M_1 c_1 q + M_2 c_2 p \pmod{n} ,$$

where $c_1 = q^{-1} \pmod{p}$ and $c_2 = p^{-1} \pmod{q}$. This gives

$$M = M_1 (q^{-1} \bmod p) q + M_2 (p^{-1} \bmod q) p \pmod{n} .$$

In order to prove this, we simply show that

$$\begin{aligned} M \pmod{p} &= M_1 \cdot 1 + 0 = M_1 , \\ M \pmod{q} &= 0 + M_2 \cdot 1 = M_2 . \end{aligned}$$

The MRC algorithm, on the other hand, computes the final number u by first computing a triangular table of values:

$$\begin{array}{lllll} u_{11} & & & & \\ u_{21} & u_{22} & & & \\ u_{31} & u_{32} & u_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ u_{k1} & u_{k2} & \cdots & \cdots & u_{k,k} \end{array}$$

where the first column of the values u_{i1} are the given values of u_i, i.e., $u_{i1} = u_i$. The values in the remaining columns are computed sequentially using the values from the previous column according to the recursion

$$u_{i,j+1} = (u_{ij} - u_{jj})c_{ji} \pmod{p_i} ,$$

where c_{ji} is the multiplicative inverse of p_j modulo p_i, i.e.,

$$c_{ji}p_j = 1 \pmod{p_i} .$$

For example, u_{32} is computed as

$$u_{32} = (u_{31} - u_{11})c_{13} \pmod{p_3} ,$$

where c_{13} is the inverse of p_1 modulo p_3. The final value of u is computed using the summation

$$u = u_{11} + u_{22}p_1 + u_{33}p_1p_2 + \cdots + u_{kk}p_1p_2\cdots p_{k-1}$$

which does not require a final modulo P reduction. Applying the MRC algorithm to the RSA decryption, we first compute

$$\begin{aligned} M_1 &:= C^{d_1} \pmod{p} , \\ M_2 &:= C^{d_2} \pmod{q} , \end{aligned}$$

where d_1 and d_2 are the same as before. The triangular table in this case is rather small, and consists of

$$\begin{array}{ll} M_{11} & \\ M_{21} & M_{22} \end{array}$$

where $M_{11} = M_1$, $M_{21} = M_2$, and

$$M_{22} = (M_{21} - M_{11})(p^{-1} \bmod q) \pmod{q} .$$

Therefore, M is computed using

$$M := M_1 + [(M_2 - M_1) \cdot (p^{-1} \bmod q) \bmod q] \cdot p .$$

This expression is correct since

$$\begin{aligned} M \pmod{p} &= M_1 + 0 = M_1 , \\ M \pmod{q} &= M_1 + (M_2 - M_1) \cdot 1 = M_2 . \end{aligned}$$

The MRC algorithm is more advantageous than the SRC algorithm for two reasons:

- It requires a single inverse computation: $p^{-1} \bmod q$.
- It does not require the final modulo n reduction.

The inverse value ($p^{-1} \bmod q$) can be precomputed and saved. Here, we note that the order of p and q in the summation in the proposed public-key cryptography standard PKCS # 1 is the reverse of our notation. The data structure [43] holding the values of user's private key has the variables:

```
exponent1 INTEGER, -- d mod (p-1)
exponent2 INTEGER, -- d mod (q-1)
coefficient INTEGER, -- (inverse of q) mod p
```

Thus, it uses ($q^{-1} \bmod p$) instead of ($p^{-1} \bmod q$). Let M_1 and M_2 be defined as before. By reversing p, q and M_1, M_2 in the summation, we obtain

$$M := M_2 + [(M_1 - M_2) \cdot (q^{-1} \bmod p) \bmod p] \cdot q \ .$$

This summation is also correct since

$$\begin{aligned} M \pmod q &= M_2 + 0 = M_2 \ , \\ M \pmod p &= M_2 + (M_1 - M_2) \cdot 1 = M_1 \ , \end{aligned}$$

as required. Assuming p and q are $(k/2)$-bit binary numbers, and d is as large as n which is a k-bit integer, we now calculate the total number of bit operations for the RSA decryption using the MRC algorithm. Assuming d_1, d_2, ($p^{-1} \bmod q$) are precomputed, and that the exponentiation algorithm is the binary method, we calculate the required number of multiplications as

- Computation of M_1: $\frac{3}{2}(k/2)$ $(k/2)$-bit multiplications.
- Computation of M_2: $\frac{3}{2}(k/2)$ $(k/2)$-bit multiplications.
- Computation of M: One $(k/2)$-bit subtraction, two $(k/2)$-bit multiplications, and one k-bit addition.

Also assuming multiplications are of order k^2, and subtractions are of order k, we calculate the total number of bit operations as

$$2\,\frac{3k}{4}(k/2)^2 + 2(k/2)^2 + (k/2) + k \ = \ \frac{3k^3}{8} + \frac{k^2 + 3k}{2} \ .$$

On the other hand, the algorithm without the CRT would compute $M = C^d \pmod n$ directly, using $(3/2)k$ k-bit multiplications which require $3k^3/2$ bit operations. Thus, considering the high-order terms, we conclude that the CRT based algorithm will be approximately 4 times faster.

4.2 Improving Montgomery's Method

The Montgomery method uses the Montgomery multiplication algorithm in order to compute multiplications and squarings required during the exponentiation process. One drawback of the algorithm is that it requires the computation of n' which has the property

$$r \cdot r^{-1} - n \cdot n' = 1 \ ,$$

where $r = 2^k$ and the k-bit number n is the RSA modulus. In this section, we show how to speed up the computation of n' within the MonPro routine. Our first observation is that we do not need the entire value of n'. We repeat the MonPro routine from Section 3.8 in order to explain this observation:

function MonPro($\bar{a}, \bar{b}$)
Step 1. $t := \bar{a} \cdot \bar{b}$
Step 2. $m := t \cdot n' \bmod r$
Step 3. $u := (t + m \cdot n)/r$
Step 4. **if** $u \geq n$ **then return** $u - n$
else return u

The multiplication of these multi-precision numbers are performed by breaking them into words, as shown in Section 3.2. Let w be the wordsize of the computer. Then, these large numbers can be thought of integers represented in radix $W = 2^w$. Assuming, these numbers require s words in their radix W representation, we can take $r = 2^{sw}$. The multiplication routine, then, accomplishes its task by computing a series of inner-product operations. For example, the multiplication of $\bar{a}$ and $\bar{b}$ in Step 1 is performed using:

1. **for** $i = 0$ **to** $s - 1$
2. $\quad C := 0$
3. $\quad$ **for** $j = 0$ **to** $s - 1$
4. $\quad\quad (C, S) := t_{i+j} + \bar{a}_j \cdot \bar{b}_i + C$
5. $\quad\quad t_{i+j} := S$
6. $\quad t_{i+s} := C$

When $\bar{a} = \bar{b}$, we can use the squaring algorithm given in Section 3.5. This will provide about 50 % savings in the time spent in Step 1 of the MonPro routine. The final value obtained is the $2s$-precision integer $(t_{2s-1}t_{2s-2}\cdots t_0)$. The computation of m and u in Steps 2 and 3 of the MonPro routine can be interleaved. We first take $u = t$, and then add $m \cdot n$ to it using the standard multiplication routine, and finally divide it by 2^{sw} which is accomplished using a shift operation (or, we just ignore the lower sw bits of u). Since $m = t \cdot n' \bmod r$ and the interleaving process proceeds word by word, we can use $n'_0 = n' \bmod 2^w$ instead of n'. This observation was made by Dussé and Kaliski [10], and used in their RSA implementation for the Motorola DSP 56000.

Thus, after t is computed by multiplying $\bar{a}$ and $\bar{b}$ using the above code, we proceed with the following code which updates t in order to compute $t + m \cdot n$.

7. for $i = 0$ to $s-1$
8. $\quad C := 0$
9. $\quad m := t_i \cdot n'_0 \bmod 2^w$
10. $\quad$ for $j = 0$ to $s-1$
11. $\quad\quad (C, S) := t_{i+j} + m \cdot n_j + C$
12. $\quad\quad t_{i+j} := S$
13. $\quad$ for $j = i+s$ to $2s-1$
14. $\quad\quad (C, S) := t_j + C$
15. $\quad\quad t_j := S$
16. $t_{2s} := C$

In Step 9, we multiply t_i by n'_0 modulo 2^w to compute m. This value of m is then used in the inner-product step. Steps 13, 14, and 15 are needed to take of the carry propagating to the last word of t. We did not need these steps in multiplying $\bar{a}$ and $\bar{b}$ (Steps 1–6) since the initial value of t was zero. In Step 16, we save the last carry out of the operation in Step 14. Thus, the length of the variable t becomes $2s+1$ due to this carry. After Step 16, we divide t by r, i.e., simply ignore the lower half of t. The resulting value is u which is then compared to n; if it is larger than n, we subtract n from it and return this value. These steps of the MonPro routine are given below:

17. for $j = 0$ to s
18. $\quad u_j := t_{j+s}$
19. $B = 0$
20. for $j = 0$ to s
21. $\quad (B, D) := u_j - n_j - B$
22. $\quad v_j := D$
23. if $B = 0$ **then return** $(v_{s-1} v_{s-2} \cdots v_0)$
$\quad\quad$ **else return** $(u_{s-1} u_{s-2} \cdots u_0)$

Thus, we have greatly simplified the MonPro routine by avoiding the full computation of n', and by using only single-precision multiplication to multiply t and n'. In the following, we will give an efficient algorithm for computing n'_0. However, before that, we give an example in which the computations performed in the MonPro routine are summarized. In this example, we will use decimal arithmetic for simplicity of the illustration. Let $n = 311$ and $r = 1000$. It is easy to show that the inverse of r is

$$r^{-1} = 65 \pmod{n} ,$$

and also that

$$n' = \frac{r \cdot r^{-1} - 1}{n} = \frac{1000 \cdot 65 - 1}{311} = 209 ,$$

and thus, $n'_0 = 9$. We will compute the Montgomery product of 216 and 123, which is equal to 248 since

$$\text{MonPro}(216, 123) = 216 \cdot 123 \cdot r^{-1} = 248 \pmod{n} .$$

The first step of the algorithm is to compute the product $216 \cdot 123$, accomplished in Steps 1–6. The initial value of t is zero, i.e., $t = 000\ 000$.

i	j	(C, S)	$t = 000\ 000$
0	0	$0 + 6 \cdot 3 + 0 = 18$	000 008
	1	$0 + 1 \cdot 3 + 1 = 04$	000 048
	2	$0 + 2 \cdot 3 + 0 = 06$	000 648
1	0	$4 + 6 \cdot 2 + 0 = 16$	000 668
	1	$6 + 1 \cdot 2 + 1 = 09$	000 968
	2	$0 + 2 \cdot 2 + 0 = 04$	004 968
2	0	$9 + 6 \cdot 1 + 0 = 15$	004 568
	1	$4 + 1 \cdot 1 + 1 = 06$	006 568
	2	$0 + 2 \cdot 1 + 0 = 02$	026 568

Then, we execute Steps 7 through 16, in order to compute $(t + m \cdot n)$ using the value of $n'_0 = 9$. The initial value of $t = 026\ 568$ comes from the previous step. Steps 7 through 16 are illustrated below:

i	m mod 10	j	(C, S)	$t = 026\ 568$
0	$8 \cdot 9 = 2$	0	$8 + 2 \cdot 1 + 0 = 10$	026 560
		1	$6 + 2 \cdot 1 + 1 = 09$	026 590
		2	$5 + 2 \cdot 3 + 0 = 11$	026 190
		3	$6 + 1 = 07$	027 190
		4	$2 + 0 = 02$	027 190
		5	$0 + 0 = 00$	027 190
1	$9 \cdot 9 = 1$	0	$9 + 1 \cdot 1 + 0 = 10$	027 100
		1	$1 + 1 \cdot 1 + 1 = 03$	027 300
		2	$7 + 1 \cdot 3 + 0 = 10$	020 300
		4	$2 + 1 = 03$	030 300
		5	$0 + 0 = 00$	030 300
2	$3 \cdot 9 = 7$	0	$3 + 7 \cdot 1 + 0 = 10$	030 000
		1	$0 + 7 \cdot 1 + 1 = 08$	038 000
		2	$3 + 7 \cdot 3 + 0 = 24$	048 000
		5	$0 + 2 = 02$	0 248 000

After Step 15, we divide t by r by shifting it s words to the right. Thus, we obtain the value of u as 248. Then, subtraction is performed to check if $u \geq n$; if it is, $u - n$ is returned as the final product value. Since in our example $248 < 311$, we return 248 as the result of the routine MonPro(126, 123), which is the correct value.

As we have pointed out earlier, there is an efficient algorithm for computing the single precision integer n'_0. The computation of n'_0 can be performed by a specialized Euclidean algorithm instead of the general extended Euclidean algorithm. Since $r = 2^{sw}$ and

$$r \cdot r^{-1} - n \cdot n' = 1 \ ,$$

we take modulo 2^w of the both sides, and obtain

$$-n \cdot n' = 1 \pmod{2^w} ,$$

or, in other words,

$$n'_0 = -n_0^{-1} \pmod{2^w} ,$$

where n'_0 and n_0^{-1} are the least significant words (the least significant w bits) of n' and n^{-1}, respectively. In order to compute $-n_0^{-1} \pmod{2^w}$, we use the algorithm given below which computes $x^{-1} \pmod{2^w}$ for a given odd x.

function ModInverse($x, 2^w$) { x is odd }
1. $y_1 := 1$
2. **for** $i = 2$ **to** w
3. **if** $2^{i-1} < x \cdot y_{i-1} \pmod{2^i}$
then $y_i := y_{i-1} + 2^{i-1}$
else $y_i := y_{i-1}$
4. **return** y_w

The correctness of the algorithm follows from the observation that, at every step i, we have

$$x \cdot y_i = 1 \pmod{2^i} .$$

This algorithm is very efficient, and uses single precision addition and multiplications in order to compute x^{-1}. As an example, we compute $23^{-1} \pmod{64}$ using the above algorithm. Here we have $x = 23$, $w = 6$. The steps of the algorithm, the temporary values, and the final inverse are shown below:

i	2^i	y_{i-1}	$x \cdot y_{i-1} \pmod{2^i}$	2^{i-1}	y_i
2	4	1	$23 \cdot 1 = 3$	2	$1 + 2 = 3$
3	8	3	$23 \cdot 3 = 5$	4	$3 + 4 = 7$
4	16	7	$23 \cdot 7 = 1$	8	7
5	32	7	$23 \cdot 7 = 1$	16	7
6	64	7	$23 \cdot 7 = 33$	32	$7 + 32 = 39$

Thus, we compute $23^{-1} = 39 \pmod{64}$. This is indeed the correct value since

$$23 \cdot 39 = 14 \cdot 64 + 1 = 1 \pmod{64} .$$

Also, at every step i, we have $x \cdot y_i = 1 \pmod{2^i}$, as shown below:

i	$x \cdot y_i \bmod 2^i$
1	$23 \cdot 1 = 1 \bmod 2$
2	$23 \cdot 3 = 1 \bmod 4$
3	$23 \cdot 7 = 1 \bmod 8$
4	$23 \cdot 7 = 1 \bmod 16$
5	$23 \cdot 7 = 1 \bmod 32$
6	$23 \cdot 39 = 1 \bmod 64$

4.3 Performance Analysis

In this section, we give timing analyses of the RSA encryption and decryption operations. This analysis can be used to estimate the performance of the RSA encryption and decryption operations on a given computer system. The analysis is based on the following assumptions.

Algorithmic Issues:

1. The exponentiation algorithm is the binary method.
2. The Montgomery reduction algorithm is used for the modular multiplications.
3. The improvements on the Montgomery method are taken into account.

Data Size:

1. The size of n is equal to s words.
2. The sizes of p and q are $s/2$ words.
3. The sizes of M and C are s words.
4. The size of e is k_e bits.
5. The Hamming weight of e is equal to h_e, where $1 < h_e \leq k_e$.
6. The size of d is k_d bits.
7. The Hamming weight of d is equal to h_d, where $1 < h_d \leq k_d$.

Precomputed Values:

1. The private exponents d_1 and d_2 are precomputed and available.
2. The coefficient $(p^{-1} \bmod q)$ or $(q^{-1} \bmod p)$ is precomputed and available.

Computer Platform:

1. The wordsize of the computer is w bits.
2. The addition of two single-precision integers requires A cycles.
3. The multiplication of two single-precision integers requires P cycles.
4. The inner-product operation requires $2A + P$ cycles.

In the following sections, we will analyze the performance of the RSA encryption and decryptions operations separately based on the preceding assumptions.

4.3.1 RSA Encryption

The encryption operation using the Montgomery product first computes n_0', which requires

$$\sum_{j=2}^{w}(P+A)=(w-1)(P+A) \tag{4.1}$$

cycles. It then proceeds to compute $\bar{M}=M\cdot r \pmod n$ and $\bar{C}=1\cdot r \pmod n$. The computation of $\bar{M}$ requires sw s-precision subtractions. The computation of $\bar{C}$, on the other hand, may require up to w s-precision subtractions. Thus, these operations together require

$$sw(sA)+w(sA)=(s^2+s)wA \tag{4.2}$$

cycles. We then start the exponentiation algorithm which requires (k_e-1) Montgomery square and (h_e-1) Montgomery product operations. The Montgomery product operation first computes the product $\bar{a}\cdot\bar{b}$ which requires

$$\sum_{i=0}^{s-1}\sum_{j=0}^{s-1}(P+2A)=s^2(P+2A)$$

cycles. Then, Steps 7 through 15 are followed, requiring

$$\sum_{i=0}^{s-1}\left[P+\sum_{j=0}^{s-1}(P+2A)+\sum_{j=i+s}^{2s-1}A\right]=sP+s^2(P+2A)+\frac{s^2+s}{2}A=(s^2+s)P+\frac{5s^2+s}{2}A$$

cycles. The s-precision subtraction operation which is performed in Steps 18–21 requires a total of s single-precision subtractions. Thus, Steps 7 through 22 require a total of

$$(s^2+s)P+\frac{5s^2+s}{2}A+sA=(s^2+s)P+\frac{5s^2+3s}{2}A$$

Thus, we calculate the total number of cycles required by the Montgomery product routine as

$$s^2(P+2A)+(s^2+s)P+\frac{5s^2+3s}{2}A=(2s^2+s)P+\frac{9s^2+3s}{2}A\ . \tag{4.3}$$

The Montgomery square routine uses the optimized squaring algorithm of Section 3.5 in order to compute $\bar{a}\cdot\bar{a}$. This step requires

$$\frac{s(s-1)}{2}(P+2A)$$

cycles. The remainder of the Montgomery square algorithm is the same as the Montgomery product algorithm. Thus, the Montgomery square routine requires a total of

$$\frac{s(s-1)}{2}(P+2A)+(s^2+s)P+\frac{5s^2+3s}{2}A=\frac{3s^2+s}{2}P+\frac{7s^2+s}{2}A \tag{4.4}$$

cycles. The total number of cycles required by the RSA encryption operation is then found by adding the number of cycles for computing n_0' given by Equation (4.1), the number of cycles required by computing $\bar{M}$ and $\bar{C}$ given by Equation (4.2), $(k_e - 1)$ times the number of cycles required by the Montgomery square operation given by Equation (4.4), and $(h_e - 1)$ times the number cycles required by the Montgomery product operation given by Equation (4.3). The total number of cycles is found as

$$\begin{aligned} T_1(s, k_e, h_e, w, P, A) &= (w-1)(P+A) + (s^2+s)wA + (k_e - 1)\left[\frac{3s^2+s}{2}P + \frac{7s^2+s}{2}A\right] \\ &+ (h_e - 1)\left[(2s^2+s)P + \frac{9s^2+3s}{2}A\right] . \end{aligned} \quad (4.5)$$

4.3.2 RSA Decryption without the CRT

The RSA decryption operation without the Chinese remainder theorem by disregarding the knowledge of the factors of the user's modulus is the same operation as the RSA encryption. Thus, the total number of cycles required by the RSA decryption operation is the same as the one given in Equation (4.5), except that k_e and h_e are replaced by k_d and h_d, respectively.

$$\begin{aligned} T_1(s, k_d, h_d, w, P, A) &= (w-1)(P+A) + (s^2+s)wA + (k_d - 1)\left[\frac{3s^2+s}{2}P + \frac{7s^2+s}{2}A\right] \\ &+ (h_d - 1)\left[(2s^2+s)P + \frac{9s^2+3s}{2}A\right] . \end{aligned} \quad (4.6)$$

4.3.3 RSA Decryption with the CRT

The RSA decryption operation using the Chinese remainder theorem first computes M_1 and M_2 using

$$\begin{aligned} M_1 &:= C^{d_1} \pmod{p} , \\ M_2 &:= C^{d_2} \pmod{q} . \end{aligned}$$

The computation of M_1 is equivalent to the RSA encryption with the exponent d_1 and modulus p. Assuming the number of words required to represent p is equal to $s/2$, we find the number of cycles required in computing M_1 as

$$T_1(\frac{s}{2}, k_{d_1}, h_{d_1}, w, P, A) ,$$

where k_{d_1} and h_{d_1} is the bit size and Hamming weight of d_1, respectively. Similarly the computation of M_2 requires

$$T_1(\frac{s}{2}, k_{d_2}, h_{d_2}, w, P, A) .$$

cycles. Then, the mixed-radix conversion algorithm computes M using

$$M := M_1 + (M_2 - M_1) \cdot (p^{-1} \bmod q) \cdot p \ ,$$

which requires one $s/2$-precision subtraction, two s-precision multiplications, and one s-precision addition. This requires a total of

$$\frac{s}{2}A + 2s^2(P + 2A) + sA = 2s^2P + (4s^2 + \frac{3s}{2})A$$

cycles assuming the coefficient $(p^{-1} \bmod q)$ is available. Therefore, we compute the total number of cycles required by the RSA decryption operation with the CRT as

$$\begin{aligned} T_2(s, k_{d_1}, h_{d_1}, k_{d_2}, h_{d_2}, w, P, A) &= T_1(\frac{s}{2}, k_{d_1}, h_{d_1}, w, P, A) + T_1(\frac{s}{2}, k_{d_2}, h_{d_2}, w, P, A) \\ &\quad + 2s^2P + (4s^2 + \frac{3s}{2})A \ . \end{aligned} \tag{4.7}$$

4.3.4 Simplified Analysis

In this section, we will consider three cases in order to simplify the performance analysis of the RSA encryption and decryption operations.

Short Exponent RSA Encryption: We will take the public exponent as $e = 2^{16} + 1$. Thus, $k_e = 17$ and $h_e = 2$. This gives the total number of cycles as

$$\begin{aligned} T_{es}(s, w, P, A) &= \left[Aw + 26P + \frac{121A}{2}\right] s^2 + \left[Aw + 9P + \frac{19A}{2}\right] s + \\ &\quad + (w - 1)(P + A) \ . \end{aligned} \tag{4.8}$$

Long Exponent RSA Encryption: We will assume that the public exponent has exactly k bits (i.e., the number of bits in n), and its Hamming weight is equal to $k/2$. Thus, $k_e = k = sw$ and $h_e = k/2 = sw/2$. This case is also equivalent to the RSA decryption without the CRT in terms of the number of cycles required to perform the operation. This gives the total number of cycles as

$$\begin{aligned} T_{el}(s, w, P, A) &= \left[\frac{5Pw}{2} + \frac{23Aw}{4}\right] s^3 + \left[Pw + \frac{9Aw}{4} - \frac{7P}{2} - 8A\right] s^2 + \\ &\quad + \left[Aw - \frac{3P}{2} - 2A\right] s + (w - 1)(P + A) \ . \end{aligned} \tag{4.9}$$

RSA Decryption with CRT: The number of bits and the Hamming weights of d_1 and d_2 are assumed to be given as $k_{d_1} = k_{d_2} = k/2 = sw/2$ and $h_{d_1} = h_{d_2} = k/4 = sw/4$. Since $k_{d_1} = k_{d_2}$ and $h_{d_1} = h_{d_2}$, we have

$$T_{dl}(s, w, P, A) = 2T_1(\frac{s}{2}, \frac{sw}{2}, \frac{sw}{4}, w, P, A) + 2s^2P + (4s^2 + \frac{3s}{2})A \ .$$

Substituting $k_{d_1} = sw/2$ and $h_{d_1} = sw/4$, we obtain

$$T_{dl}(s,w,P,A) = \left[\frac{5Pw}{8} + \frac{23Aw}{16}\right]s^3 + \left[\frac{Pw}{2} + \frac{9Aw}{8} + \frac{P}{4}\right]s^2 + + \left[Aw - \frac{3P}{2} - \frac{A}{2}\right]s + 2(w-1)(P+A) \ . \qquad (4.10)$$

4.3.5 An Example

In a given computer implementation, the values of w, P, and A are fixed. Thus, the number of cycles required is a function of s, i.e., the word-length of the modulus. In this section, we will apply the above analysis to the Analog Devices Signal Processor ADSP 2105. This signal processor has a data path of $w = 16$ bits, and runs with a clock speed of 10 MHz. Furthermore, examining the arithmetic instructions, we have determined that the ADSP 2105 signal processor adds or multiplies two single-precision numbers in a single clock cycle. Considering the read and write times, we take $A = 3$ and $P = 3$. The simplified expressions for T_{es}, T_{el}, and T_{dl} are given below:

$$\begin{aligned} T_{es} &= \frac{615}{2}s^2 + \frac{207}{2}s + 90 \ , \\ T_{el} &= 396s^3 + \frac{243}{2}s^2 + \frac{75}{2}s + 90 \ , \\ T_{dl} &= 99s^3 + \frac{315}{4}s^2 + 42s + 180 \ . \end{aligned}$$

Using the clock cycle time of the ADSP 2105 as 100 ns, we tabulate the encryption and decryption times for the values of $k = 128, 256, 384, \ldots, 1024$, corresponding to the values of $s = 8, 16, 24, \ldots, 64$, respectively. The following table summarizes the times (in milliseconds) of the short exponent RSA encryption (T_{es}), the long exponent RSA encryption (T_{el}), and the RSA decryption with the CRT (T_{dl}).

k	T_{es}	T_{el}	T_{dl}
128	3	21	6
256	8	165	43
384	18	555	142
512	32	1,310	333
640	50	2,554	646
768	71	4,408	1,113
896	97	6,993	1,764
1024	127	10,431	2,628

Our experiments with the ADSP simulator validated these estimated values. However, we note that the values of P and A must be carefully determined for a reliable estimation of the timings of the RSA encryption and decryption operations.

[25] Ç. K. Koç and C. Y. Hung. Multi-operand modulo addition using carry save adders. *Electronics Letters*, 26(6):361–363, 15th March 1990.

[26] Ç. K. Koç and C. Y. Hung. Bit-level systolic arrays for modular multiplication. *Journal of VLSI Signal Processing*, 3(3):215–223, 1991.

[27] Ç. K. Koç and C. Y. Hung. Adaptive m-ary segmentation and canonical recoding algorithms for multiplication of large binary numbers. *Computers and Mathematics with Applications*, 24(3):3–12, 1992.

[28] M. Kochanski. Developing an RSA chip. In H. C. Williams, editor, *Advances in Cryptology — CRYPTO 85, Proceedings*, Lecture Notes in Computer Science, No. 218, pages 350–357. New York, NY: Springer-Verlag, 1985.

[29] I. Koren. *Computer Arithmetic Algorithms.* Englewood Cliffs, NJ: Prentice-Hall, 1993.

[30] D. Laurichesse and L. Blain. Optimized implementation of RSA cryptosystem. *Computers & Security*, 10(3):263–267, May 1991.

[31] J. D. Lipson. *Elements of Algebra and Algebraic Computing.* Reading, MA: Addison-Wesley, 1981.

[32] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[33] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. Rapport de Recherche 983, INRIA, March 1989.

[34] National Institute for Standards and Technology. Digital signature standard (DSS). *Federal Register*, 56:169, August 1991.

[35] J. Olivos. On vectorial addition chains. *Journal of Algorithms*, 2(1):13–21, March 1981.

[36] J. M. Pollard. The fast Fourier transform in a finite field. *Mathematics of Computation*, 25:365–374, 1971.

[37] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, 18(21):905–907, October 1982.

[38] G. W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.

[39] H. Riesel. *Prime Numbers and Computer Methods for Factorization.* Boston, MA: Birkhäuser, 1985.

[40] R. L. Rivest. RSA chips (Past/Present/Future). In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology, Proceedings of EUROCRYPT 84*, Lecture Notes in Computer Science, No. 209, pages 159–165. New York, NY: Springer-Verlag, 1984.

[41] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[42] RSA Laboratories. Answers to Frequently Asked Questions About Today's Cryptography. RSA Data Security, Inc., October 1993.

[43] RSA Laboratories. The Public-Key Cryptography Standards (PKCS). RSA Data Security, Inc., November 1993.

[44] A. Schönhage. A lower bound for the length of addition chains. *Theoretical Computer Science*, 1:1–12, 1975.

[45] A. Schönhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971.

[46] H. Sedlak. The RSA cryptography processor. In D. Chaum and W. L. Price, editors, *Advances in Cryptology — EUROCRYPT 87*, Lecture Notes in Computer Science, No. 304, pages 95–105. New York, NY: Springer-Verlag, 1987.

[47] K. R. Sloan, Jr. Comments on "A computer algorithm for the product AB modulo M". *IEEE Transactions on Computers*, 34(3):290–292, March 1985.

[48] E. E. Swartzlander, editor. *Computer Arithmetic*, volume I. Los Alamitos, CA: IEEE Computer Society Press, 1990.

[49] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and its Applications to Computer Technology*. New York, NY: McGraw-Hill, 1967.

[50] C. D. Walter. Systolic modular multiplication. *IEEE Transactions on Computers*, 42(3):376–378, March 1993.

[51] S. Waser and M. J. Flynn. *Introduction to Arithmetic for Digital System Designers*. New York, NY: Holt, Rinehart and Winston, 1982.

[52] Y. Yacobi. Exponentiating faster with addition chains. In I. B. Damgård, editor, *Advances in Cryptology — EUROCRYPT 90*, Lecture Notes in Computer Science, No. 473, pages 222–229. New York, NY: Springer-Verlag, 1990.

[53] A. C.-C. Yao. On the evaluation of powers. *SIAM Journal on Computing*, 5(1):100–103, March 1976.

Rivest

Express Mail Mailing Label No. EL636302715[illegible]

CD

Programming Techniques

S.L. Graham, R.L. Rivest*
Editors

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R. L. Rivest, A. Shamir, and L. Adleman
MIT Laboratory for Computer Science and Department of Mathematics

An encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption key. This has two important consequences:
(1) Couriers or other secure means are not needed to transmit keys, since a message can be enciphered using an encryption key publicly revealed by the intended recipient. Only he can decipher the message, since only he knows the corresponding decryption key.
(2) A message can be "signed" using a privately held decryption key. Anyone can verify this signature using the corresponding publicly revealed encryption key. Signatures cannot be forged, and a signer cannot later deny the validity of his signature. This has obvious applications in "electronic mail" and "electronic funds transfer" systems. A message is encrypted by representing it as a number M, raising M to a publicly specified power e, and then taking the remainder when the result is divided by the publicly specified product, n, of two large secret prime numbers p and q. Decryption is similar; only a different, secret, power d is used, where $e * d \equiv 1 (\text{mod } (p - 1) * (q - 1))$. The security of the system rests in part on the difficulty of factoring the published divisor, n.

Key Words and Phrases: digital signatures, public-key cryptosystems, privacy, authentication, security, factorization, prime number, electronic mail, message-passing, electronic funds transfer, cryptography.
CR Categories: 2.12, 3.15, 3.50, 3.81, 5.25

This research was supported by National Science Foundation grant MCS76-14294, and the Office of Naval Research grant number N00014-67-A-0204-0063.

* Note. This paper was submitted prior to the time that Rivest became editor of the department, and editorial consideration was completed under the former editor, G. K. Manacher.

Authors' Address: MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139.

I. Introduction

The era of "electronic mail" [10] may soon be upon us; we must ensure that two important properties of the current "paper mail" system are preserved: (a) messages are *private*, and (b) messages can be *signed*. We demonstrate in this paper how to build these capabilities into an electronic mail system.

At the heart of our proposal is a new encryption method. This method provides an implementation of a "public-key cryptosystem", an elegant concept invented by Diffie and Hellman [1]. Their article motivated our research, since they presented the concept but not any practical implementation of such a system. Readers familiar with [1] may wish to skip directly to Section V for a description of our method.

II. Public-Key Cryptosystems

In a "public-key cryptosystem" each user places in a public file an encryption procedure E. That is, the public file is a directory giving the encryption procedure of each user. The user keeps secret the details of his corresponding decryption procedure D. These procedures have the following four properties:

(a) Deciphering the enciphered form of a message M yields M. Formally,

$$D(E(M)) = M. \tag{1}$$

(b) Both E and D are easy to compute.

(c) By publicly revealing E the user does not reveal an easy way to compute D. This means that in practice only he can decrypt messages encrypted with E, or compute D efficiently.

(d) If a message M is first deciphered and then enciphered, M is the result. Formally,

$$E(D(M)) = M. \tag{2}$$

An encryption (or decryption) procedure typically consists of a *general method* and an *encryption key*. The general method, under control of the key, enciphers a message M to obtain the enciphered form of the message, called the *ciphertext* C. Everyone can use the same general method; the security of a given procedure will rest on the security of the key. Revealing an encryption algorithm then means revealing the key.

When the user reveals E he reveals a very *inefficient* method of computing D(C): testing all possible messages M until one such that E(M) = C is found. If property (c) is satisfied the number of such messages to test will be so large that this approach is impractical.

A function E satisfying (a)-(c) is a "trap-door one-way function;" if it also satisfies (d) it is a "trap-door one-way permutation." Diffie and Hellman [1] introduced the concept of trap-door one-way functions but

did not present any examples. These functions are called "one-way" because they are easy to compute in one direction but (apparently) very difficult to compute in the other direction. They are called "trap-door" functions since the inverse functions are in fact easy to compute once certain private "trap-door" information is known. A trap-door one-way function which also satisfies (d) must be a permutation: every message is the ciphertext for some other message and every ciphertext is itself a permissible message. (The mapping is "one-to-one" and "onto"). Property (d) is needed only to implement "signatures".

The reader is encouraged to read Diffie and Hellman's excellent article [1] for further background, for elaboration of the concept of a public-key cryptosystem, and for a discussion of other problems in the area of cryptography. The ways in which a public-key cryptosystem can ensure privacy and enable "signatures" (described in Sections III and IV below) are also due to Diffie and Hellman.

For our scenarios we suppose that A and B (also known as Alice and Bob) are two users of a public-key cryptosystem. We will distinguish their encryption and decryption procedures with subscripts: E_A, D_A, E_B, D_B.

III. Privacy

Encryption is the standard means of rendering a communication private. The sender enciphers each message before transmitting it to the receiver. The receiver (but no unauthorized person) knows the appropriate deciphering function to apply to the received message to obtain the original message. An eavesdropper who hears the transmitted message hears only "garbage" (the ciphertext) which makes no sense to him since he does not know how to decrypt it.

The large volume of personal and sensitive information currently held in computerized data banks and transmitted over telephone lines makes encryption increasingly important. In recognition of the fact that efficient, high-quality encryption techniques are very much needed but are in short supply, the National Bureau of Standards has recently adopted a "Data Encryption Standard" [13, 14], developed at IBM. The new standard does not have property (c), needed to implement a public-key cryptosystem.

All classical encryption methods (including the NBS standard) suffer from the "key distribution problem." The problem is that before a private communication can begin, *another* private transaction is necessary to distribute corresponding encryption and decryption keys to the sender and receiver, respectively. Typically a private courier is used to carry a key from the sender to the receiver. Such a practice is not feasible if an electronic mail system is to be rapid and inexpensive. A public-key cryptosystem needs no private couriers; the keys can be distributed over the insecure communications channel.

How can Bob send a private message M to Alice in a public-key cryptosystem? First, he retrieves E_A from the public file. Then he sends her the enciphered message $E_A(M)$. Alice deciphers the message by computing $D_A(E_A(M)) = M$. By property (c) of the public-key cryptosystem only she can decipher $E_A(M)$. She can encipher a private response with E_B, also available in the public file.

Observe that no private transactions between Alice and Bob are needed to establish private communication. The only "setup" required is that each user who wishes to receive private communications must place his enciphering algorithm in the public file.

Two users can also establish private communication over an insecure communications channel without consulting a public file. Each user sends his encryption key to the other. Afterwards all messages are enciphered with the encryption key of the recipient, as in the public-key system. An intruder listening in on the channel cannot decipher any messages, since it is not possible to derive the decryption keys from the encryption keys. (We assume that the intruder cannot modify or insert messages into the channel.) Ralph Merkle has developed another solution [5] to this problem.

A public-key cryptosystem can be used to "bootstrap" into a standard encryption scheme such as the NBS method. Once secure communications have been established, the first message transmitted can be a key to use in the NBS scheme to encode all following messages. This may be desirable if encryption with our method is slower than with the standard scheme. (The NBS scheme is probably somewhat faster if special-purpose hardware encryption devices are used; our scheme may be faster on a general-purpose computer since multiprecision arithmetic operations are simpler to implement than complicated bit manipulations.)

IV. Signatures

If electronic mail systems are to replace the existing paper mail system for business transactions, "signing" an electronic message must be possible. The recipient of a signed message has proof that the message originated from the sender. This quality is stronger than mere authentication (where the recipient can verify that the message came from the sender); the recipient can convince a "judge" that the signer sent the message. To do so, he must convince the judge that he did not forge the signed message himself! In an authentication problem the recipient does not worry about this possibility, since he only wants to satisfy *himself* that the message came from the sender.

An electronic signature must be *message*-dependent, as well as *signer*-dependent. Otherwise the recipient could modify the message before showing the message-signature pair to a judge. Or he could attach the signature to any message whatsoever, since it is impossible to detect electronic "cutting and pasting."

To implement signatures the public-key cryptosys-

tem must be implemented with trap-door one-way permutations (i.e. have property (d)), since the decryption algorithm will be applied to unenciphered messages.

How can user Bob send Alice a "signed" message M in a public-key cryptosystem? He first computes his "signature" S for the message M using D_B:

$$S = D_B(M).$$

(Deciphering an unenciphered message "makes sense" by property (d) of a public key cryptosystem: each message is the ciphertext for some other message.) He then encrypts S using E_A (for privacy), and sends the result $E_A(S)$ to Alice. He need not send M as well; it can be computed from S.

Alice first decrypts the ciphertext with D_A to obtain S. She knows who is the presumed sender of the signature (in this case, Bob); this can be given if necessary in plain text attached to S. She then extracts the message with the encryption procedure of the sender, in this case E_B (available on the public file):

$$M = E_B(S).$$

She now possesses a message-signature pair (M, S) with properties similar to those of a signed paper document.

Bob cannot later deny having sent Alice this message, since no one else could have created $S = D_B(M)$. Alice can convince a "judge" that $E_B(S) = M$, so she has proof that Bob signed the document.

Clearly Alice cannot modify M to a different version M', since then she would have to create the corresponding signature $S' = D_B(M')$ as well.

Therefore Alice has received a message "signed" by Bob, which she can "prove" that he sent, but which she cannot modify. (Nor can she forge his signature for any other message.)

An electronic checking system could be based on a signature system such as the above. It is easy to imagine an encryption device in your home terminal allowing you to sign checks that get sent by electronic mail to the payee. It would only be necessary to include a unique check number in each check so that even if the payee copies the check the bank will only honor the first version it sees.

Another possibility arises if encryption devices can be made fast enough: it will be possible to have a telephone conversation in which every word spoken is signed by the encryption device before transmission.

When encryption is used for signatures as above, it is important that the encryption device not be "wired in" between the terminal (or computer) and the communications channel, since a message may have to be successively enciphered with several keys. It is perhaps more natural to view the encryption device as a "hardware subroutine" that can be executed as needed.

We have assumed above that each user can always access the public file reliably. In a "computer network" this might be difficult; an "intruder" might forge messages purporting to be from the public file. The user would like to be sure that he actually obtains the encryption procedure of his desired correspondent and not, say, the encryption procedure of the intruder. This danger disappears if the public file "signs" each message it sends to a user. The user can check the signature with the public file's encryption algorithm E_{PF}. The problem of "looking up" E_{PF} itself in the public file is avoided by giving each user a description of E_{PF} when he first shows up (in person) to join the public-key cryptosystem and to deposit his public encryption procedure. He then stores this description rather than ever looking it up again. The need for a courier between every pair of users has thus been replaced by the requirement for a single secure meeting between each user and the public-file manager when the user joins the system. Another solution is to give each user, when he signs up, a book (like a telephone directory) containing all the encryption keys of users in the system.

V. Our Encryption and Decryption Methods

To encrypt a message M with our method, using a public encryption key (e, n), proceed as follows. (Here e and n are a pair of positive integers.)

First, represent the message as an integer between 0 and $n - 1$. (Break a long message into a series of blocks, and represent each block as such an integer.) Use any standard representation. The purpose here is not to encrypt the message but only to get it into the numeric form necessary for encryption.

Then, encrypt the message by raising it to the eth power modulo n. That is, the result (the ciphertext C) is the remainder when M^e is divided by n.

To decrypt the ciphertext, raise it to another power d, again modulo n. The encryption and decryption algorithms E and D are thus:

$$C \equiv E(M) \equiv M^e \pmod{n}, \text{ for a message M.}$$
$$D(C) \equiv C^d \pmod{n}, \text{ for a ciphertext C.}$$

Note that encryption does not increase the size of a message; both the message and the ciphertext are integers in the range 0 to $n - 1$.

The *encryption key* is thus the pair of positive integers (e, n). Similarly, the *decryption key* is the pair of positive integers (d, n). Each user makes his encryption key public, and keeps the corresponding decryption key private. (These integers should properly be subscripted as in n_A, e_A, and d_A, since each user has his own set. However, we will only consider a typical set, and will omit the subscripts.)

How should you choose your encryption and decryption keys, if you want to use our method?

You first compute n as the product of two primes p and q:

$$n = p * q.$$

These primes are very large, "random" primes. Al-

though you will make n public, the factors p and q will be effectively hidden from everyone else due to the enormous difficulty of factoring n. This also hides the way d can be derived from e.

You then pick the integer d to be a large, random integer which is relatively prime to $(p - 1) * (q - 1)$. That is, check that d satisfies:

$\gcd(d, (p - 1) * (q - 1)) = 1$
("gcd" means "greatest common divisor").

The integer e is finally computed from p, q, and d to be the "multiplicative inverse" of d, modulo $(p - 1) * (q - 1)$. Thus we have

$e * d \equiv 1 \pmod{(p - 1) * (q - 1)}$.

We prove in the next section that this guarantees that (1) and (2) hold, i.e. that E and D are inverse permutations. Section VII shows how each of the above operations can be done efficiently.

The aforementioned method should not be confused with the "exponentiation" technique presented by Diffie and Hellman [1] to solve the key distribution problem. Their technique permits two users to determine a key in common to be used in a normal cryptographic system. It is not based on a trap-door one-way permutation. Pohlig and Hellman [8] study a scheme related to ours, where exponentiation is done modulo a prime number.

VI. The Underlying Mathematics

We demonstrate the correctness of the deciphering algorithm using an identity due to Euler and Fermat [7]: for any integer (message) M which is relatively prime to n,

$$M^{\varphi(n)} \equiv 1 \pmod{n}. \qquad (3)$$

Here $\varphi(n)$ is the Euler totient function giving the number of positive integers less than n which are relatively prime to n. For prime numbers p,

$\varphi(p) = p - 1$.

In our case, we have by elementary properties of the totient function [7]:

$$\begin{aligned} \varphi(n) &= \varphi(p) * \varphi(q), \\ &= (p - 1) * (q - 1) \\ &= n - (p + q) + 1. \end{aligned} \qquad (4)$$

Since d is relatively prime to $\varphi(n)$, it has a multiplicative inverse e in the ring of integers modulo $\varphi(n)$:

$$e * d \equiv 1 \pmod{\varphi(n)}. \qquad (5)$$

We now prove that equations (1) and (2) hold (that is, that deciphering works correctly if e and d are chosen as above). Now

$D(E(M)) \equiv (E(M))^d \equiv (M^e)^d \equiv M^{e \cdot d} \pmod{n}$

$E(D(M)) \equiv (D(M))^e \equiv (M^d)^e \equiv M^{e \cdot d} \pmod{n}$

and

$M^{e \cdot d} \equiv M^{k \cdot \varphi(n)+1} \pmod{n}$ (for some integer k).

From (3) we see that for all M such that p does not divide M

$M^{p-1} \equiv 1 \pmod{p}$

and since $(p - 1)$ divides $\varphi(n)$

$M^{k \cdot \varphi(n)+1} \equiv M \pmod{p}$.

This is trivially true when $M \equiv 0 \pmod{p}$, so that this equality actually holds for *all* M. Arguing similarly for q yields

$M^{k \cdot \varphi(n)+1} \equiv M \pmod{q}$.

Together these last two equations imply that for all M,

$M^{e \cdot d} \equiv M^{k \cdot \varphi(n)+1} \equiv M \pmod{n}$.

This implies (1) and (2) for all M, $0 \le M < n$. Therefore E and D are inverse permutations. (We thank Rich Schroeppel for suggesting the above improved version of the authors' previous proof.)

VII. Algorithms

To show that our method is practical, we describe an efficient algorithm for each required operation.

A. How to Encrypt and Decrypt Efficiently

Computing $M^e \pmod{n}$ requires at most $2 * \log_2(e)$ multiplications and $2 * \log_2(e)$ divisions using the following procedure (decryption can be performed similarly using d instead of e):

Step 1. Let $e_k e_{k-1} \ldots e_1 e_0$ be the binary representation of e.
Step 2. Set the variable C to 1.
Step 3. Repeat steps 3a and 3b for $i = k, k - 1, \ldots, 0$:
  Step 3a. Set C to the remainder of C^2 when divided by n.
  Step 3b. If $e_i = 1$, then set C to the remainder of $C * M$ when divided by n.
Step 4. Halt. Now C is the encrypted form of M.

This procedure is called "exponentiation by repeated squaring and multiplication." This procedure is half as good as the best; more efficient procedures are known. Knuth [3] studies this problem in detail.

The fact that the enciphering and deciphering are identical leads to a simple implementation. (The whole operation can be implemented on a few special-purpose integrated circuit chips.)

A high-speed computer can encrypt a 200-digit message M in a few seconds; special-purpose hardware would be much faster. The encryption time per block increases no faster than the cube of the number of digits in n.

B. How to Find Large Prime Numbers

Each user must (privately) choose two large ran-

dom prime numbers p and q to create his own encryption and decryption keys. These numbers must be large so that it is not computationally feasible for anyone to factor $n = p * q$. (Remember that n, but not p or q, will be in the public file.) We recommend using 100-digit (decimal) prime numbers p and q, so that n has 200 digits.

To find a 100-digit "random" prime number, generate (odd) 100-digit random numbers until a prime number is found. By the prime number theorem [7], about $(\ln 10^{100})/2 = 115$ numbers will be tested before a prime is found.

To test a large number b for primality we recommend the elegant "probabilistic" algorithm due to Solovay and Strassen [12]. It picks a random number a from a uniform distribution on $\{1, \ldots, b-1\}$, and tests whether

$$\gcd(a, b) = 1 \text{ and } J(a, b) \equiv a^{(b-1)/2} \pmod{b}, \tag{6}$$

where $J(a, b)$ is the Jacobi symbol [7]. If b is prime (6) is always true. If b is composite (6) will be false with probability at least $1/2$. If (6) holds for 100 randomly chosen values of a then b is almost certainly prime; there is a (negligible) chance of one in 2^{100} that b is composite. Even if a composite were accidentally used in our system, the receiver would probably detect this by noticing that decryption didn't work correctly. When b is odd, $a \le b$, and $\gcd(a, b) = 1$, the Jacobi symbol $J(a, b)$ has a value in $\{-1, 1\}$ and can be efficiently computed by the program:

$$
\begin{aligned}
J(a, b) = \; & \textbf{if } a = 1 \textbf{ then } 1 \textbf{ else} \\
& \textbf{if } a \text{ is even } \textbf{then } J(a/2, b) * (-1)^{(b^2-1)/8} \\
& \textbf{else } J(b \pmod{a}, a) * (-1)^{(a-1)*(b-1)/4}
\end{aligned}
$$

(The computations of $J(a, b)$ and $\gcd(a, b)$ can be nicely combined, too.) Note that this algorithm does *not* test a number for primality by trying to factor it. Other efficient procedures for testing a large number for primality are given in [6, 9, 11].

To gain additional protection against sophisticated factoring algorithms, p and q should differ in length by a few digits, both $(p-1)$ and $(q-1)$ should contain large prime factors, and $\gcd(p-1, q-1)$ should be small. The latter condition is easily checked.

To find a prime number p such that $(p-1)$ has a large prime factor, generate a large random prime number u, then let p be the first prime in the sequence $i * u + 1$, for $i = 2, 4, 6, \ldots$. (This shouldn't take too long.) Additional security is provided by ensuring that $(u-1)$ also has a large prime factor.

A high-speed computer can determine in several seconds whether a 100-digit number is prime, and can find the first prime after a given point in a minute or two.

Another approach to finding large prime numbers is to take a number of known factorization, add one to it, and test the result for primality. If a prime p is found it is possible to *prove* that it really is prime by using the factorization of $p - 1$. We omit a discussion of this since the probabilistic method is adequate.

C. How to Choose d

It is very easy to choose a number d which is relatively prime to $\varphi(n)$. For example, any prime number greater than $\max(p, q)$ will do. It is important that d should be chosen from a large enough set so that a cryptanalyst cannot find it by direct search.

D. How to Compute e from d and $\varphi(n)$

To compute e, use the following variation of Euclid's algorithm for computing the greatest common divisor of $\varphi(n)$ and d. (See exercise 4.5.2.15 in [3].) Calculate $\gcd(\varphi(n), d)$ by computing a series $x_0, x_1, x_2, \ldots$, where $x_0 = \varphi(n)$, $x_1 = d$, and $x_{i+1} \equiv x_{i-1} \pmod{x_i}$, until an x_k equal to 0 is found. Then $\gcd(x_0, x_1) = x_{k-1}$. Compute for each x_i numbers a_i and b_i such that $x_i = a_i * x_0 + b_i * x_1$. If $x_{k-1} = 1$ then b_{k-1} is the multiplicative inverse of $x_1 \pmod{x_0}$. Since k will be less than $2 * \log_2(n)$, this computation is very rapid.

If e turns out to be less than $\log_2(n)$, start over by choosing another value of d. This guarantees that every encrypted message (except $M = 0$ or $M = 1$) undergoes some "wrap-around" (reduction modulo n).

VIII. A Small Example

Consider the case $p = 47$, $q = 59$, $n = p * q = 47 * 59 = 2773$, and $d = 157$. Then $\varphi(2773) = 46 * 58 = 2668$, and e can be computed as follows:

$x_0 = 2668$,	$a_0 = 1$,	$b_0 = 0$,
$x_1 = 157$,	$a_1 = 0$,	$b_1 = 1$,
$x_2 = 156$,	$a_2 = 1$,	$b_2 = -16$ (since $2668 = 157 * 16 + 156$),
$x_3 = 1$,	$a_3 = -1$,	$b_3 = 17$ (since $157 = 1 * 156 + 1$).

Therefore $e = 17$, the multiplicative inverse (mod 2668) of $d = 157$.

With $n = 2773$ we can encode two letters per block, substituting a two-digit number for each letter: blank = 00, A = 01, B = 02, ..., Z = 26. Thus the message

ITS ALL GREEK TO ME

(Julius Caesar, I, ii, 288, paraphrased) is encoded:

0920 1900 0112 1200 0718
0505 1100 2015 0013 0500

Since $e = 10001$ in binary, the first block ($M = 920$) is enciphered:

$$M^{17} \equiv (((((1)^2 * M)^2)^2)^2)^2 * M \equiv 948 \pmod{2773}.$$

The whole message is enciphered as:

0948 2342 1084 1444 2663
2390 0778 0774 0219 1655.

The reader can check that deciphering works: $948^{157} \equiv 920 \pmod{2773}$, etc.

IX. Security of the Method: Cryptanalytic Approaches

Since no techniques exist to *prove* that an encryption scheme is secure, the only test available is to see whether anyone can think of a way to break it. The NBS standard was "certified" this way; seventeen man-years at IBM were spent fruitlessly trying to break that scheme. Once a method has successfully resisted such a concerted attack it may for practical purposes be considered secure. (Actually there is some controversy concerning the security of the NBS method [2].)

We show in the next sections that all the obvious approaches for breaking our system are at least as difficult as factoring n. While factoring large numbers is not provably difficult, it is a well-known problem that has been worked on for the last three hundred years by many famous mathematicians. Fermat (1601?–1665) and Legendre (1752–1833) developed factoring algorithms; some of today's more efficient algorithms are based on the work of Legendre. As we shall see in the next section, however, no one has yet found an algorithm which can factor a 200-digit number in a reasonable amount of time. We conclude that our system has already been partially "certified" by these previous efforts to find efficient factoring algorithms.

In the following sections we consider ways a cryptanalyst might try to determine the secret decryption key from the publicly revealed encryption key. We do not consider ways of protecting the decryption key from theft; the usual physical security methods should suffice. (For example, the encryption device could be a separate device which could also be used to *generate* the encryption and decryption keys, such that the decryption key is never printed out (even for its owner) but only used to decrypt messages. The device could erase the decryption key if it was tampered with.)

A. Factoring n

Factoring n would enable an enemy cryptanalyst to "break" our method. The factors of n enable him to compute $\varphi(n)$ and thus d. Fortunately, factoring a number seems to be much more difficult than determining whether it is prime or composite.

A large number of factoring algorithms exist. Knuth [3, Section 4.5.4] gives an excellent presentation of many of them. Pollard [9] presents an algorithm which factors a number n in time $O(n^{1/4})$.

The fastest factoring algorithm known to the authors is due to Richard Schroeppel (unpublished); it can factor n in approximately

$$\begin{aligned} &\exp(\mathrm{sqrt}(\ln(n) * \ln(\ln(n)))) \\ &= n^{\mathrm{sqrt}(\ln(\ln(n))/\ln(n))} \\ &= (\ln(n))^{\mathrm{sqrt}(\ln(n)/\ln(\ln(n)))} \end{aligned}$$

steps (here ln denotes the natural logarithm function). Table I gives the number of operations needed to factor n with Schroeppel's method, and the time required if each operation uses one microsecond, for various lengths of the number n (in decimal digits):

Table I.

Digits	Number of operations	Time
50	1.4×10^{10}	3.9 hours
75	9.0×10^{12}	104 days
100	2.3×10^{15}	74 years
200	1.2×10^{23}	3.8×10^{9} years
300	1.5×10^{29}	4.9×10^{15} years
500	1.3×10^{39}	4.2×10^{25} years

We recommend that n be about 200 digits long. Longer or shorter lengths can be used depending on the relative importance of encryption speed and security in the application at hand. An 80-digit n provides moderate security against an attack using current technology; using 200 digits provides a margin of safety against future developments. This flexibility to choose a key-length (and thus a level of security) to suit a particular application is a feature not found in many of the previous encryption schemes (such as the NBS scheme).

B. Computing $\varphi(n)$ Without Factoring n

If a cryptanalyst could compute $\varphi(n)$ then he could break the system by computing d as the multiplicative inverse of e modulo $\varphi(n)$ (using the procedure of Section VII D).

We argue that this approach is no easier than factoring n since it enables the cryptanalyst to easily factor n using $\varphi(n)$. This approach to factoring n has not turned out to be practical.

How can n be factored using $\varphi(n)$? First, $(p + q)$ is obtained from n and $\varphi(n) = n - (p + q) + 1$. Then $(p - q)$ is the square root of $(p + q)^2 - 4n$. Finally, q is half the difference of $(p + q)$ and $(p - q)$.

Therefore breaking our system by computing $\varphi(n)$ is no easier than breaking our system by factoring n. (This is why n must be composite; $\varphi(n)$ is trivial to compute if n is prime.)

C. Determining d Without Factoring n or Computing $\varphi(n)$.

Of course, d should be chosen from a large enough set so that a direct search for it is unfeasible.

We argue that computing d is no easier for a cryptanalyst than factoring n, since once d is known n could be factored easily. This approach to factoring has also not turned out to be fruitful.

A knowledge of d enables n to be factored as follows. Once a cryptanalyst knows d he can calculate $e * d - 1$, which is a multiple of $\varphi(n)$. Miller [6] has shown that n can be factored using any multiple of $\varphi(n)$. Therefore if n is large a cryptanalyst should not be able to determine d any easier than he can factor n.

A cryptanalyst may hope to find a d' which is equivalent to the d secretly held by a user of the

public-key cryptosystem. If such values d' were common then a brute-force search could break the system. However, all such d' differ by the least common multiple of $(p-1)$ and $(q-1)$, and finding one enables n to be factored. (In (3) and (5), $\varphi(n)$ can be replaced by $\mathrm{lcm}(p-1, q-1)$.) Finding any such d' is therefore as difficult as factoring n.

D. Computing D in Some Other Way

Although this problem of "computing eth roots modulo n without factoring n" is not a well-known difficult problem like factoring, we feel reasonably confident that it is computationally intractable. It may be possible to prove that any general method of breaking our scheme yields an efficient factoring algorithm. This would establish that any way of breaking our scheme must be as difficult as factoring. We have not been able to prove this conjecture, however.

Our method should be certified by having the above conjecture of intractability withstand a concerted attempt to disprove it. The reader is challenged to find a way to "break" our method.

X. Avoiding "Reblocking" when Encrypting a Signed Message

A signed message may have to be "reblocked" for encryption since the signature n may be larger than the encryption n (every user has his own n). This can be avoided as follows. A threshold value h is chosen (say $h = 10^{199}$) for the public-key cryptosystem. Every user maintains *two* public (e, n) pairs, one for enciphering and one for signature-verification, where every signature n is less than h, and every enciphering n is greater than h. Reblocking to encipher a signed message is then unnecessary; the message is blocked according to the transmitter's signature n.

Another solution uses a technique given in [4]. Each user has a single (e, n) pair where n is between h and $2h$, where h is a threshold as above. A message is encoded as a number less than h and enciphered as before, except that if the ciphertext is greater than h, it is repeatedly re-enciphered until it is less than h. Similarly for decryption the ciphertext is repeatedly deciphered to obtain a value less than h. If n is near h re-enciphering will be infrequent. (Infinite looping is not possible, since at worst a message is enciphered as itself.)

XI. Conclusions

We have proposed a method for implementing a public-key cryptosystem whose security rests in part on the difficulty of factoring large numbers. If the security of our method proves to be adequate, it permits secure communications to be established without the use of couriers to carry keys, and it also permits one to "sign" digitized documents.

The security of this system needs to be examined in more detail. In particular, the difficulty of factoring large numbers should be examined very closely. The reader is urged to find a way to "break" the system. Once the method has withstood all attacks for a sufficient length of time it may be used with a reasonable amount of confidence.

Our encryption function is the only candidate for a "trap-door one-way permutation" known to the authors. It might be desirable to find other examples, to provide alternative implementations should the security of our system turn out someday to be inadequate. There are surely also many new applications to be discovered for these functions.

Acknowledgments. We thank Martin Hellman, Richard Schroeppel, Abraham Lempel, and Roger Needham for helpful discussions, and Wendy Glasser for her assistance in preparing the initial manuscript. Xerox PARC provided support and some marvelous text-editing facilities for preparing the final manuscript.

Received April 4, 1977; revised September 1, 1977

References
1. Diffie, W., and Hellman, M. New directions in cryptography. *IEEE Trans. Inform. Theory IT-22*, 6 (Nov. 1976), 644–654.
2. Diffie, W., and Hellman, M. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer 10* (June 1977), 74–84.
3. Knuth, D. E. *The Art of Computer Programming, Vol 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Mass., 1969.
4. Levine, J., and Brawley, J.V. Some cryptographic applications of permutation polynomials. *Cryptologia 1* (Jan. 1977), 76–92.
5. Merkle, R. Secure communications over an insecure channel. Submitted to *Comm. ACM*.
6. Miller, G.L. Riemann's hypothesis and tests for primality. Proc. Seventh Annual ACM Symp. on the Theory of Comptng. Albuquerque, New Mex., May 1975, pp. 234–239; extended vers. available as Res. Rep. CS-75-27, Dept. of Comptr. Sci., U. of Waterloo, Waterloo, Ont., Canada, Oct. 1975.
7. Niven, I., and Zuckerman, H.S. *An Introduction to the Theory of Numbers*. Wiley, New York, 1972.
8. Pohlig, S.C., and Hellman, M.E. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. To appear in IEEE Trans. Inform. Theory, 1978.
9. Pollard, J.M. Theorems on factorization and primality testing. *Proc. Camb. Phil. Soc. 76* (1974), 521–528.
10. Potter, R.J., Electronic mail. *Science 195*, 4283 (March 1977), 1160–1164.
11. Rabin, M.O., Probabilistic algorithms. In *Algorithms and Complexity*, J. F. Traub, Ed., Academic Press, New York, 1976, pp. 21–40.
12. Solovay, R., and Strassen, V. A Fast Monte-Carlo test for primality. *SIAM J. Comptng. 6* (March 1977), 84–85.
13. *Federal Register, Vol. 40*, No. 52, March 17, 1975.
14. *Federal Register, Vol. 40*, No. 149, August 1, 1975.

(A comment on this article may be found in the Technical Correspondence section of this issue, page 173.—Ed.)

RSA Labs

Express Mail Mailing Label No.: EL636302715US

CE

PKCS #1: RSA Encryption Standard

An RSA Laboratories Technical Note
Version 1.5
Revised November 1, 1993*

1. Scope

This standard describes a method for encrypting data using the RSA public-key cryptosystem. Its intended use is in the construction of digital signatures and digital envelopes, as described in PKCS #7:

- For digital signatures, the content to be signed is first reduced to a message digest with a message-digest algorithm (such as MD5), and then an octet string containing the message digest is encrypted with the RSA private key of the signer of the content. The content and the encrypted message digest are represented together according to the syntax in PKCS #7 to yield a digital signature. This application is compatible with Privacy-Enhanced Mail (PEM) methods.

- For digital envelopes, the content to be enveloped is first encrypted under a content-encryption key with a content-encryption algorithm (such as DES), and then the content-encryption key is encrypted with the RSA public keys of the recipients of the content. The encrypted content and the encrypted content-encryption key are represented together according to the syntax in PKCS #7 to yield a digital envelope. This application is also compatible with PEM methods.

The standard also describes a syntax for RSA public keys and private keys. The public-key syntax would be used in certificates; the private-key syntax would be used typically in PKCS #8 private-key information. The public-key syntax is identical to that in both

*Supersedes June 3, 1991 version, which was also published as NIST/OSI Implementors' Workshop document SEC-SIG-91-18. PKCS documents are available by electronic mail to <pkcs@rsa.com>.

003-903018-150-000-000

X.509 and Privacy-Enhanced Mail. Thus X.509/PEM RSA keys can be used in this standard.

The standard also defines three signature algorithms for use in signing X.509/PEM certificates and certificate-revocation lists, PKCS #6 extended certificates, and other objects employing digital signatures such as X.401 message tokens.

Details on message-digest and content-encryption algorithms are outside the scope of this standard, as are details on sources of the pseudorandom bits required by certain methods in this standard.

2. References

FIPS PUB 46–1 National Bureau of Standards. *FIPS PUB 46–1: Data Encryption Standard.* January 1988.

PKCS #6 RSA Laboratories. *PKCS #6: Extended-Certificate Syntax Standard.* Version 1.5, November 1993.

PKCS #7 RSA Laboratories. *PKCS #7: Cryptographic Message Syntax Standard.* Version 1.5, November 1993.

PKCS #8 RSA Laboratories. *PKCS #8: Private-Key Information Syntax Standard.* Version 1.2, November 1993.

RFC 1319 B. Kaliski. *RFC 1319: The MD2 Message-Digest Algorithm.* April 1992.

RFC 1320 R. Rivest. *RFC 1320: The MD4 Message-Digest Algorithm.* April 1992.

RFC 1321 R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* April 1992.

RFC 1423 D. Balenson. *RFC 1423: Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers.* February 1993.

X.208 CCITT. *Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1).* 1988.

X.209 CCITT. *Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).* 1988.

X.411 CCITT. *Recommendation X.411: Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures.* 1988.

X.509 CCITT. *Recommendation X.509: The Directory—Authentication Framework.* 1988.

[dBB92] B. den Boer and A. Bosselaers. An attack on the last two rounds of MD4. In J. Feigenbaum, editor, *Advances in Cryptology—CRYPTO '91 Proceedings,* volume 576 of *Lecture Notes in Computer Science,* pages 194–203. Springer-Verlag, New York, 1992.

[dBB93] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. Presented at EUROCRYPT '93 (Lofthus, Norway, May 24–27, 1993).

[DO86] Y. Desmedt and A.M. Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In H.C. Williams, editor, *Advances in Cryptology—CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 516–521. Springer-Verlag, New York, 1986.

[Has88] Johan Hastad. Solving simultaneous modular equations. *SIAM Journal on Computing*, 17(2):336–341, April 1988.

[IM90] Colin I'Anson and Chris Mitchell. Security defects in CCITT Recommendation X.509—The directory authentication framework. *Computer Communications Review*, :30–34, April 1990.

[Mer90] R.C. Merkle. Note on MD4. Unpublished manuscript, 1990.

[Mil76] G.L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and Systems Sciences*, 13(3):300–307, 1976.

[QC82] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, 18(21):905–907, October 1982.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

3. Definitions

For the purposes of this standard, the following definitions apply.

`AlgorithmIdentifier`: A type that identifies an algorithm (by object identifier) and associated parameters. This type is defined in X.509.

ASN.1: Abstract Syntax Notation One, as defined in X.208.

BER: Basic Encoding Rules, as defined in X.209.

DES: Data Encryption Standard, as defined in FIPS PUB 46-1.

MD2: RSA Data Security, Inc.'s MD2 message-digest algorithm, as defined in RFC 1319.

MD4: RSA Data Security, Inc.'s MD4 message-digest algorithm, as defined in RFC 1320.

MD5: RSA Data Security, Inc.'s MD5 message-digest algorithm, as defined in RFC 1321.

modulus: Integer constructed as the product of two primes.

PEM: Internet Privacy-Enhanced Mail, as defined in RFC 1423 and related documents.

RSA: The RSA public-key cryptosystem, as defined in [RSA78].

private key: Modulus and private exponent.

public key: Modulus and public exponent.

4. Symbols and abbreviations

Upper-case italic symbols (e.g., *BT*) denote octet strings and bit strings (in the case of the signature *S*); lower-case italic symbols (e.g., *c*) denote integers.

ab	hexadecimal octet value	c	exponent
BT	block type	d	private exponent
D	data	e	public exponent
EB	encryption block	k	length of modulus in octets
ED	encrypted data	n	modulus
M	message	p, q	prime factors of modulus
MD	message digest	x	integer encryption block
MD'	comparative message digest	y	integer encrypted data
PS	padding string	mod n	modulo n
S	signature	$X \| Y$	concatenation of X, Y
$\|\|X\|\|$ length in octets of X			

5. General overview

The next six sections specify key generation, key syntax, the encryption process, the decryption process, signature algorithms, and object identifiers.

Each entity shall generate a pair of keys: a public key and a private key. The encryption process shall be performed with one of the keys and the decryption process shall be performed with the other key. Thus the encryption process can be either a public-key operation or a private-key operation, and so can the decryption process. Both processes transform an octet string to another octet string. The processes are inverses of each other if one process uses an entity's public key and the other process uses the same entity's private key.

The encryption and decryption processes can implement either the classic RSA transformations, or variations with padding.

6. Key generation

This section describes RSA key generation.

Each entity shall select a positive integer e as its public exponent.

Each entity shall privately and randomly select two distinct odd primes p and q such that $(p-1)$ and e have no common divisors, and $(q-1)$ and e have no common divisors.

The public modulus n shall be the product of the private prime factors p and q:

$$n = pq \, .$$

The private exponent shall be a positive integer d such that $de-1$ is divisible by both $p-1$ and $q-1$.

The length of the modulus n in octets is the integer k satisfying

$$2^{8(k-1)} \leq n < 2^{8k} \, .$$

The length k of the modulus must be at least 12 octets to accommodate the block formats in this standard (see Section 8).

Notes.

1. The public exponent may be standardized in specific applications. The values 3 and F_4 (65537) may have some practical advantages, as noted in X.509 Annex C.

2. Some additional conditions on the choice of primes may well be taken into account in order to deter factorization of the modulus. These security conditions fall outside the scope of this standard. The lower bound on the length k is to accommodate the block formats, not for security.

7. Key syntax

This section gives the syntax for RSA public and private keys.

7.1 Public-key syntax

An RSA public key shall have ASN.1 type `RSAPublicKey`:

```
RSAPublicKey ::= SEQUENCE {
  modulus INTEGER, -- n
  publicExponent INTEGER -- e }
```

(This type is specified in X.509 and is retained here for compatibility.)

The fields of type `RSAPublicKey` have the following meanings:

- `modulus` is the modulus n.
- `publicExponent` is the public exponent e.

7.2 Private-key syntax

An RSA private key shall have ASN.1 type `RSAPrivateKey`:

```
RSAPrivateKey ::= SEQUENCE {
  version Version,
  modulus INTEGER, -- n
  publicExponent INTEGER, -- e
  privateExponent INTEGER, -- d
  prime1 INTEGER, -- p
  prime2 INTEGER, -- q
  exponent1 INTEGER, -- d mod (p-1)
  exponent2 INTEGER, -- d mod (q-1)
  coefficient INTEGER -- (inverse of q) mod p }

Version ::= INTEGER
```

The fields of type `RSAPrivateKey` have the following meanings:

- `version` is the version number, for compatibility with future revisions of this standard. It shall be 0 for this version of the standard.
- `modulus` is the modulus n.
- `publicExponent` is the public exponent e.
- `privateExponent` is the private exponent d.
- `prime1` is the prime factor p of n.
- `prime2` is the prime factor q of n.
- `exponent1` is $d \bmod (p-1)$.
- `exponent2` is $d \bmod (q-1)$.

- `coefficient` is the Chinese Remainder Theorem coefficient q^{-1} mod p.

Notes.

1. An RSA private key logically consists of only the modulus n and the private exponent d. The presence of the values p, q, d mod $(p-1)$, d mod $(p-1)$, and q^{-1} mod p is intended for efficiency, as Quisquater and Couvreur have shown [QC82]. A private-key syntax that does not include all the extra values can be converted readily to the syntax defined here, provided the public key is known, according to a result by Miller [Mil76].

2. The presence of the public exponent e is intended to make it straightforward to derive a public key from the private key.

8. Encryption process

This section describes the RSA encryption process.

The encryption process consists of four steps: encryption-block formatting, octet-string-to-integer conversion, RSA computation, and integer-to-octet-string conversion. The input to the encryption process shall be an octet string D, the data; an integer n, the modulus; and an integer c, the exponent. For a public-key operation, the integer c shall be an entity's public exponent e; for a private-key operation, it shall be an entity's private exponent d. The output from the encryption process shall be an octet string ED, the encrypted data.

The length of the data D shall not be more than k–11 octets, which is positive since the length k of the modulus is at least 12 octets. This limitation guarantees that the length of the padding string PS is at least eight octets, which is a security condition.

Notes.

1. In typical applications of this standard to encrypt content-encryption keys and message digests, one would have $||D|| \leq 30$. Thus the length of the RSA modulus will need to be at least 328 bits (41 octets), which is reasonable and consistent with security recommendations.

2. The encryption process does not provide an explicit integrity check to facilitate error detection should the encrypted data be corrupted in transmission. However, the structure of the encryption block guarantees that the probability that corruption is undetected is less than 2^{-16}, which is

an upper bound on the probability that a random encryption block looks like block type `02`.

3. Application of private-key operations as defined here to data other than an octet string containing a message digest is not recommended and is subject to further study.

4. This standard may be extended to handle data of length more than k-11 octets.

8.1 Encryption-block formatting

A block type *BT*, a padding string *PS*, and the data *D* shall be formatted into an octet string *EB*, the encryption block.

$$EB = 00 \,||\, BT \,||\, PS \,||\, 00 \,||\, D\,. \tag{1}$$

The block type *BT* shall be a single octet indicating the structure of the encryption block. For this version of the standard it shall have value `00`, `01`, or `02`. For a private-key operation, the block type shall be `00` or `01`. For a public-key operation, it shall be `02`.

The padding string *PS* shall consist of $k-3-||D||$ octets. For block type `00`, the octets shall have value `00`; for block type `01`, they shall have value `FF`; and for block type `02`, they shall be pseudorandomly generated and nonzero. This makes the length of the encryption block *EB* equal to k.

Notes.

1. The leading `00` octet ensures that the encryption block, converted to an integer, is less than the modulus.

2. For block type `00`, the data *D* must begin with a nonzero octet or have known length so that the encryption block can be parsed unambiguously. For block types `01` and `02`, the encryption block can be parsed unambiguously since the padding string *PS* contains no octets with value `00` and the padding string is separated from the data *D* by an octet with value `00`.

3. Block type `01` is recommended for private-key operations. Block type `01` has the property that the encryption block, converted to an integer, is guaranteed to be large, which prevents certain attacks of the kind proposed by Desmedt and Odlyzko [DO86].

4. Block types `01` and `02` are compatible with PEM RSA encryption of content-encryption keys and message digests as described in RFC 1423.

5. For block type `02`, it is recommended that the pseudorandom octets be generated independently for each encryption process, especially if the same data is input to more than one encryption process. Hastad's results [Has88] motivate this recommendation.

6. For block type `02`, the padding string is at least eight octets long, which is a security condition for public-key operations that prevents an attacker from recoving data by trying all possible encryption blocks. For simplicity, the minimum length is the same for block type `01`.

7. This standard may be extended in the future to include other block types.

8.2 Octet-string-to-integer conversion

The encryption block *EB* shall be converted to an integer x, the integer encryption block. Let $EB_1, \ldots, EB_k$ be the octets of *EB* from first to last. Then the integer x shall satisfy

$$x = \sum_{i=1}^{k} 2^{8(k-i)} EB_i \,. \tag{2}$$

In other words, the first octet of *EB* has the most significance in the integer and the last octet of *EB* has the least significance.

Note. The integer encryption block x satisfies $0 \le x < n$ since $EB_1 = 00$ and $2^{8(k-1)} \le n$.

8.3 RSA computation

The integer encryption block x shall be raised to the power c modulo n to give an integer y, the integer encrypted data.

$$y = x^c \bmod n, \quad 0 \le y < n \,.$$

This is the classic RSA computation.

8.4 Integer-to-octet-string conversion

The integer encrypted data y shall be converted to an octet string *ED* of length k, the encrypted data. The encrypted data *ED* shall satisfy

$$y = \sum_{i=1}^{k} 2^{8(k-i)} ED_i \,. \tag{3}$$

where $ED_1, \ldots, ED_k$ are the octets of *ED* from first to last.

In other words, the first octet of *ED* has the most significance in the integer and the last octet of *ED* has the least significance.

9. Decryption process

This section describes the RSA decryption process.

The decryption process consists of four steps: octet-string-to-integer conversion, RSA computation, integer-to-octet-string conversion, and encryption-block parsing. The input to the decryption process shall be an octet string *ED*, the encrypted data; an integer n, the modulus; and an integer c, the exponent. For a public-key operation, the integer c shall be an entity's public exponent e; for a private-key operation, it shall be an entity's private exponent d. The output from the decryption process shall be an octet string *D*, the data.

It is an error if the length of the encrypted data *ED* is not k.

For brevity, the decryption process is described in terms of the encryption process.

9.1 Octet-string-to-integer conversion

The encrypted data *ED* shall be converted to an integer y, the integer encrypted data, according to Equation (3).

It is an error if the integer encrypted data y does not satisfy $0 \le y < n$.

9.2 RSA computation

The integer encrypted data y shall be raised to the power c modulo n to give an integer x, the integer encryption block.

$$x = y^c \bmod n, \ \ 0 \le x < n .$$

This is the classic RSA computation.

9.3 Integer-to-octet-string conversion

The integer encryption block x shall be converted to an octet string *EB* of length k, the encryption block, according to Equation (2).

9.4 Encryption-block parsing

The encryption block *EB* shall be parsed into a block type *BT*, a padding string *PS*, and the data *D* according to Equation (1).

It is an error if any of the following conditions occurs:

- The encryption block *EB* cannot be parsed unambiguously (see notes to Section 8.1).
- The padding string *PS* consists of fewer than eight octets, or is inconsistent with the block type *BT*.
- The decryption process is a public-key operation and the block type *BT* is not `00` or `01`, or the decryption process is a private-key operation and the block type is not `02`.

10. Signature algorithms

This section defines three signature algorithms based on the RSA encryption process described in Sections 8 and 9. The intended use of the signature algorithms is in signing X.509/PEM certificates and certificate-revocation lists, PKCS #6 extended certificates, and other objects employing digital signatures such as X.401 message tokens. The algorithms are not intended for use in constructing digital signatures in PKCS #7. The first signature algorithm (informally, "MD2 with RSA") combines the MD2 message-digest algorithm with RSA, the second (informally, "MD4 with RSA") combines the MD4 message-digest algorithm with RSA, and the third (informally, "MD5 with RSA") combines the MD5 message-digest algorithm with RSA.

This section describes the signature process and the verification process for the two algorithms. The "selected" message-digest algorithm shall be either MD2 or MD5, depending on the signature algorithm. The signature process shall be performed with an entity's private key and the verification process shall be performed with an entity's public key. The signature process transforms an octet string (the message) to a bit string (the signature); the verification process determines whether a bit string (the signature) is the signature of an octet string (the message).

Note. The only difference between the signature algorithms defined here and one of the the methods by which signatures (encrypted message digests) are constructed in PKCS #7 is that signatures here are represented here as bit strings, for consistency with the X.509 `SIGNED` macro. In PKCS #7 encrypted message digests are octet strings.

10.1 Signature process

The signature process consists of four steps: message digesting, data encoding, RSA encryption, and octet-string-to-bit-string conversion. The input to the signature process shall be an octet string *M*, the message; and a signer's private key. The output from the signature process shall be a bit string *S*, the signature.

10.1.1 Message digesting

The message *M* shall be digested with the selected message-digest algorithm to give an octet string *MD*, the message digest.

10.1.2 Data encoding

The message digest *MD* and a message-digest algorithm identifier shall be combined into an ASN.1 value of type `DigestInfo`, described below, which shall be BER-encoded to give an octet string *D*, the data.

```
DigestInfo ::= SEQUENCE {
  digestAlgorithm DigestAlgorithmIdentifier,
  digest Digest }

DigestAlgorithmIdentifier ::= AlgorithmIdentifier

Digest ::= OCTET STRING
```

The fields of type `DigestInfo` have the following meanings:

- `digestAlgorithm` identifies the message-digest algorithm (and any associated parameters). For this application, it should identify the selected message-digest algorithm, MD2, MD4 or MD5. For reference, the relevant object identifiers are the following:

```
md2 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) US(840) rsadsi(113549)
    digestAlgorithm(2) 2 }
md4 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) US(840) rsadsi(113549)
    digestAlgorithm(2) 4 }
md5 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) US(840) rsadsi(113549)
    digestAlgorithm(2) 5 }
```

For these object identifiers, the `parameters` field of the `digestAlgorithm` value should be NULL.

- `digest` is the result of the message-digesting process, i.e., the message digest *MD*.

Notes.

1. A message-digest algorithm identifier is included in the `DigestInfo` value to limit the damage resulting from the compromise of one message-digest algorithm. For instance, suppose an adversary were able to find messages with a given MD2 message digest. That adversary might try to forge a signature on a message by finding an innocuous-looking message with the same MD2 message digest, and coercing a signer to sign the innocuous-looking message. This attack would succeed only if the signer used MD2. If the `DigestInfo` value contained only the message digest, however, an adversary could attack signers that use any message digest.

2. Although it may be claimed that the use of a `SEQUENCE` type violates the literal statement in the X.509 `SIGNED` and `SIGNATURE` macros that a signature is an `ENCRYPTED OCTET STRING` (as opposed to `ENCRYPTED SEQUENCE`), such a literal interpretation need not be required, as I'Anson and Mitchell point out [IM90].

3. No reason is known that MD4 would not be sufficient for very high security digital signature schemes, but because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. A message-digest algorithm can be considered "broken" if someone can find a collision: two messages with the same digest. While collisions have been found in variants of MD4 with only two digesting "rounds" [Mer90][dBB92], none have been found in MD4 itself, which has three rounds. After further critical review, it may be appropriate to consider MD4 for very high security applications.

   MD5, which has four rounds and is proportionally slower than MD4, is recommended until the completion of MD4's review. The reported "pseudocollisions" in MD5's internal compression function [dBB93] do not appear to have any practical impact on MD5's security.

   MD2, the slowest of the three, has the most conservative design. No attacks on MD2 have been published.

10.1.3 RSA encryption

The data *D* shall be encrypted with the signer's RSA private key as described in Section 7 to give an octet string *ED*, the encrypted data. The block type shall be `01`. (See Section 8.1.)

10.1.4 Octet-string-to-bit-string conversion

The encrypted data *ED* shall be converted into a bit string *S*, the signature. Specifically, the most significant bit of the first octet of the encrypted data shall become the first bit of the signature, and so on through the least significant bit of the last octet of the encrypted data, which shall become the last bit of the signature.

Note. The length in bits of the signature *S* is a multiple of eight.

10.2 Verification process

The verification process for both signature algorithms consists of four steps: bit-string-to-octet-string conversion, RSA decryption, data decoding, and message digesting and comparison. The input to the verification process shall be an octet string *M*, the message; a signer's public key; and a bit string *S*, the signature. The output from the verification process shall be an indication of success or failure.

10.2.1 Bit-string-to-octet-string conversion

The signature *S* shall be converted into an octet string *ED*, the encrypted data. Specifically, assuming that the length in bits of the signature *S* is a multiple of eight, the first bit of the signature shall become the most significant bit of the first octet of the encrypted data, and so on through the last bit of the signature, which shall become the least significant bit of the last octet of the encrypted data.

It is an error if the length in bits of the signature *S* is not a multiple of eight.

10.2.2 RSA decryption

The encrypted data *ED* shall be decrypted with the signer's RSA public key as described in Section 8 to give an octet string *D*, the data.

It is an error if the block type recovered in the decryption process is not `01`. (See Section 9.4.)

10.2.3 Data decoding

The data *D* shall be BER-decoded to give an ASN.1 value of type `DigestInfo`, which shall be separated into a message digest *MD* and a message-digest algorithm identifier. The message-digest algorithm identifier shall determine the "selected" message-digest algorithm for the next step.

It is an error if the message-digest algorithm identifier does not identify the MD2, MD4 or MD5 message-digest algorithm.

10.2.4 Message digesting and comparison

The message *M* shall be digested with the selected message-digest algorithm to give an octet string *MD'*, the comparative message digest. The verification process shall succeed if the comparative message digest *MD'* is the same as the message digest *MD*, and the verification process shall fail otherwise.

11. Object identifiers

This standard defines five object identifiers: `pkcs-1`, `rsaEncryption`, `md2WithRSAEncryption`, `md4WithRSAEncryption`, and `md5WithRSAEncryption`.

The object identifier `pkcs-1` identifies this standard.

```
pkcs-1 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) US(840) rsadsi(113549)
      pkcs(1) 1 }
```

The object identifier `rsaEncryption` identifies RSA public and private keys as defined in Section 7 and the RSA encryption and decryption processes defined in Sections 8 and 9.

```
rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }
```

The `rsaEncryption` object identifier is intended to be used in the `algorithm` field of a value of type `AlgorithmIdentifier`. The `parameters` field of that type, which has the algorithm-specific syntax `ANY DEFINED BY algorithm`, would have ASN.1 type `NULL` for this algorithm.

The object identifiers `md2WithRSAEncryption`, `md4WithRSAEncryption`, `md5WithRSAEncryption`, identify, respectively, the "MD2 with RSA," "MD4 with RSA," and "MD5 with RSA" signature and verification processes defined in Section 10.

```
md2WithRSAEncryption OBJECT IDENTIFIER ::= { pkcs-1 2 }
md4WithRSAEncryption OBJECT IDENTIFIER ::= { pkcs-1 3 }
md5WithRSAEncryption OBJECT IDENTIFIER ::= { pkcs-1 4 }
```

These object identifiers are intended to be used in the `algorithm` field of a value of type `AlgorithmIdentifier`. The `parameters` field of that type, which has the algorithm-specific syntax `ANY DEFINED BY algorithm`, would have ASN.1 type `NULL` for these algorithms.

Note. X.509's object identifier `rsa` also identifies RSA public keys as defined in Section 7, but does not identify private keys, and identifies different encryption and decryption

processes. It is expected that some applications will identify public keys by `rsa`. Such public keys are compatible with this standard; an `rsaEncryption` process under an `rsa` public key is the same as the `rsaEncryption` process under an `rsaEncryption` public key.

Revision history

Versions 1.0–1.3

Versions 1.0–1.3 were distributed to participants in RSA Data Security, Inc.'s Public-Key Cryptography Standards meetings in February and March 1991.

Version 1.4

Version 1.4 is part of the June 3, 1991 initial public release of PKCS. Version 1.4 was published as NIST/OSI Implementors' Workshop document SEC-SIG-91-18.

Version 1.5

Version 1.5 incorporates several editorial changes, including updates to the references and the addition of a revision history. The following substantive changes were made:

- Section 10: "MD4 with RSA" signature and verification processes are added.
- Section 11: `md4WithRSAEncryption` object identifier is added.

Author's address

RSA Laboratories
100 Marine Parkway
Redwood City, CA 94065 USA

(415) 595-7703
(415) 595-4126 (fax)
pkcs-editor@rsa.com

Express Mail NO. EL656508710US

M. O. Rabin

CF

MIT/LCS/TR-212

DIGITALIZED SIGNATURES AND PUBLIC-KEY FUNCTIONS AS INTRACTABLE AS FACTORIZATION

Michael O. Rabin

January 1979

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

DIGITALIZED SIGNATURES AND PUBLIC-KEY FUNCTIONS
AS INTRACTABLE AS FACTORIZATION

by
Michael O. Rabin
Visiting Professor of Applied Mathematics, MIT
Professor of Mathematics, Hebrew University
Jerusalem, Israel

ABSTRACT. We introduce a new class of public-key functions involving a number $n = p \cdot q$ having two large prime factors. As usual, the key n is public, while p and q are the private key used by the issuer for production of signatures and function inversion. These functions can be used for all the applications involving public-key functions proposed by Diffie and Hellman [2], including digitalized signatures. We prove that for any given n, if we can invert the function $y = E_n(x)$ for even a small percentage of the values y then we can factor n. Thus as long as factorization of large numbers remains practically intractable, for appropriatly chosen keys not even a small percentage of signatures are forgerable. Breaking the RSA function [6] is at most as hard as factorization, but is not known to be equivalent to factorization even in the weak sense that ability to invert all function values entails

ability to factor the key. Computation time for these functions, i.e. signature verification, is several hundred times faster than for the RSA scheme in [6]. Inversion time, using the private key, is comparable. The almost-everywhere intractability of signature-forgery for our functions (on the assumption that factoring is intractable) is of great practical significance and seems to be the first proved result of this kind.

Key words. Public-key functions, Digitalized signatures, Factorization, Intractable problems.

INTRODUCTION

In their fundamental paper [2] Diffie and Hellman have shown how public key trap door functions can be employed for the solution of various problems arising in electronic mail, including the production of digitalized signatures. An example of a public-key function usable for digitalized signatures was given in the elegant paper [6] by Rivest, Adelman, and Shamir, who introduced a trap-door one-way function employing a number n factorable into a product $n = p \cdot q$ of two large primes. The decoding algorithm given in [6] for this function requires knowledge of the factors p, q of n. It is, however, conceivable that another decoding algorithm exists that does not involve or imply factorization of n. Thus, breaking this one-way function is at most as difficult as factorization, but possibly easier.

We present a different public key function which can be used for digitalized signatures, and all the other applications, in the same way as the above-mentioned function. The function in [6] is 1-1. Our function is four to one, but this causes only slight modifications in the applications.

For this new function we can prove that the ability to forge signatures or decode messages is equivalent to the ability to factor large numbers. In fact, for any given n, a signature forgery or inversion algorithm effective in just a small percentage of all cases, say one case in a thousand, already leads to a factorization of n. By inversion we mean finding for a number y in the range of E <u>one</u> of the x such that $E(x) = y$.

In view of the present-day intractability of the factorization problem, this fact lends substantial support to the viability of our public-key function. As long as it is impossible in practice to factor large numbers, it will be impossible for a fixed key to forge signatures even for a small percentage of all messages.

The fact that we are able to prove, on the assumption that factoring is hard, that for our function, for a fixed key n whose factorization is not given, inversion must be hard for almost all messages is of great significance. For other trap door functions it may be the case that even though worst case complexity or even average complexity are high, in say one percent of cases inversion is

easy. From a commercial point of view this would pose an unacceptable risk. For example, an adversary can randomly search by computer for messages useful to him, such as payment instructions, on which he can forge signatures. To the best of our knowledge, we have in this article the first example of an almost everywhere difficult problem of this type.

In addition, computation time for this function is several hundred times faster, and inversion when p,q are known, is about eight times faster than the corresponding algorithms in [6]. If we invert the RSA function by Chinese Remaindering, as we do here, then inversion time for the two functions are comparable.

Theorems 1 and 2 concerning the equivalence of square-root extraction with factorization, are perhaps also of independent number-theoretic interest.

1. THE PUBLIC-KEY FUNCTION

Let $n = p \cdot q$ be the product of two large primes p,q, and let $0 \leq b < n$.

DEFINITION 1: The function $E_{n,b}(x)$ is defined for $0 \leq x < n$ by $E_{n,b}(x) \equiv x(x+b) \mod n$, $0 \leq E_{n,b}(x) < n$.

Computation of $E(x)$, for fixed n,b, requires one addition, one multiplication, and one division of

$x(x+b)$ by n to find the residue $E_{n,b}(x)$. Note that only the public key n,b, but not the factorization $n = p \cdot q$, is required for encoding.

2. INVERSION ALGORITHMS

Given $c \equiv x(x+b) \bmod n$, we want to find the four values $0 \leq x_i < n$, $1 \leq i \leq 4$ such that $E(x_i) = c$. We assume of course that the private key, i.e. the factors of n, are known.

Throughout this paper $\text{res}(A,B)$ will denote the residue of A when divided by B, and (A,B) will denote the greatest common divisor (g.c.d.) of A and B.

The decoder, who is the issuer of the public key n,b, knows the factorization $n = p \cdot q$. Clearly, it sufficies to solve the equation $x(x+b) \equiv c$ separately $\bmod p$ and $\bmod q$ and then find a solution $\bmod n$.

Let a be an integer so that $a \equiv 1 \bmod p$, $a \equiv 0 \bmod q$, and b satisfy $b \equiv 1 \bmod q$, $b \equiv \bmod p$. If r and s satisfy the congruence $\bmod p$ and $\bmod q$ respectively, then $z = ar + bs$ solves the congruence $\bmod n$, and $x = \text{res}(z,n)$ is the sought-after solution.

In what follows let p be a fixed prime. We shall understand all integers $\underline{a}$ to be residues mod p, i.e., $0 \leq a < p$. For d a quadratic residue (q.r.) mod p, $\sqrt{d}$ will denote any one of the two integers such that $(\sqrt{d})^2 \equiv$ mod p, and $-\sqrt{d}$ will denote $p - \sqrt{d}$.

To solve

$$(1) \qquad f(x) = x^2 + bx - c \equiv \text{mod } p$$

let $d = b/2$ mod p then $(x+d)^2 \equiv c + d^2$ mod p, $x = -d \pm \sqrt{c+d^2}$. We can solve the equation (1) as soon as we can extract square roots mod p, i.e., solve $y^2 - m \equiv 0$ mod p.

Assume first that $p = 4k - 1$ so that $4|(p+1)$. Since m is a q.r., $m^{\frac{p-1}{2}} \equiv 1 \text{ mod } p$. We claim that

$$(2) \qquad \ell = \sqrt{m} \equiv m^{\frac{p+1}{4}} \text{ mod } p$$

is one of the two square roots of m. Namely,

$$\ell^2 \equiv m^{\frac{p+1}{2}} \equiv m \cdot m^{\frac{p-1}{2}} \equiv m \text{ mod } p \ .$$

Thus one implementation of the function would use p and q such that $p \equiv q \equiv 3 \text{ mod } 4$, and the decoding algorithm (2).

For $p = 4k + 1$ we directly solve the equation (1) by a probabilistic algorithm. This is a special case of Berlekamp's root-finding in GF(p) algorithm given in [1].

The short proof given here is taken from [5], where generalizations to $GF(p^n)$ appear. If the roots of (1) are α, $\beta \varepsilon GF(p)$ then $x^2 + bx - c = (x - \alpha)(x - \beta)$ The roots in $GF(p)$ of the polynomial equation $x^{\frac{p-1}{2}} - 1 = 0$ are exactly the quadratic residues $\alpha \varepsilon GF(p)$. Consequently, if α is a quadratic residue while β is not, then $(x^{\frac{p-1}{2}} - 1, f(x)) = x - \alpha$, so that α and subsequently $\beta = -(b+\alpha) \bmod p$ are readily found.

Assume that α and β are of the <u>same type</u>, i.e., both quadratic residues (q.r.) or both quadratic non-residues mod p, and that $\alpha \neq \beta$. Let $0 \leq \delta < p$ then $\alpha + \delta$ and $\beta + \delta$ are of the same type if and only if $(\alpha+\delta)/(\beta+\delta)$ is a q.r. mod p. As δ takes all values $0 \leq \delta < p$ except $\delta = -\beta$, the quotient $(\alpha+\delta)/(\beta+\delta)$ takes all values $0 \leq \gamma < p$ except $\gamma = 1$. Thus for exactly $\frac{p-1}{2}$ choices δ, $\alpha+\delta$ and $\beta+\delta$ will not be of the same type.

Since $f(x-\delta) = (x-\alpha-\delta)(x-\alpha-\beta)$, we have that for a *random* choice of $0 \leq \delta < p$, with probability 1/2

$$(3) \quad (x^{\frac{p-1}{2}} - 1, f(x-\delta)) = x - \alpha - \delta \quad \text{or} \quad x - \beta - \delta.$$

Thus on the average two values of δ have to be tried for finding the roots of (1).

The computation of the g.c.d. (3) requires $O(\log_2 p)$ operations in $GF(p)$, i.e., additions and multiplications

mod p. Namely, by essentially repeated squarings starting with x, compute $x + h = \mathrm{res}(x^{\frac{p-1}{2}}, f(x-\delta))$. Whenever a quadratic polynomial is encountered, divide by $f(x-\delta)$ to produce a linear polynomial. Note that x is a formal variable so that all computations involve just pairs of residues mod p. Now by (3'), $x + h - 1$ is $x - \alpha - \delta$ or $x - \beta - \delta$, so that $-\delta - h + 1$ is a root of (1).

3. USE IN SIGNATURES

To employ E for signatures the signer P produces two large primes p,q by use of one of the prime-testing algorithms [3,7]. He forms $n = p \cdot q$, chooses a number $0 \leq b < n$ and publicizes the pair (n,b) (but not the factors p,q).

By convention, when wishing to sign a given message, M,P adds as suffix a word U of an agreed upon length k. The choice of U is randomized each time a message is to be signed. The signer now compresses $M_1 = MU$ by a hashing function to a word $C(M_1) = c$, so that as a binary number $c \leq n$; see [4]. The computation of C( ) is publicly known, so that $c = C(M_1)$ is checkable by everybody.

P now checks whether for this c the congruence

$$x(x+b) \equiv c \bmod n \qquad (4)$$

is solvable.

By the analysis of Section 2, this congruence is solvable if and only if $m = c + d^2$ is a q.r. mod p and mod q. Thus testing the solvability of (4) amounts to computing the Jacobi Symbols $(\frac{m}{p})$ and $(\frac{m}{q})$ which is essentially a g.c.d. type computation.

If congruence (4) is not solvable then P picks another random U_1 and tries $c_1 = C(MU_1)$. The expected number of tries is 4. When for some U the congruence (4) is solvable for $c = C(MU)$, P finds a solution x.

<u>DEFINITION 2</u>: For a given public key n,b used by P and an agreed upon compressing function C( ) and integer k, P's signature on a message M is a pair U,x where $\ell(U) = k$ and $x(x + b) \equiv C(MU) \bmod n$.

Anybody can check P's signature by computing $c = C(MU)$ and testing whether $x(x+b) \equiv c \bmod n$.

The randomization of the suffix U of M also adds protection against possible attacks on the function E. Without the suffix, an adversary may attempt to feed to P messages M for his signature, hoping to learn the factorization of n from the solution of $x(x+b) \equiv C(M) \bmod n$, which will be produced by P as his signature. Actually, this does not seem a serious threat because of the hashing effected by C(M).

However, the randomized suffix of length k leads to essentially 2^k possible random values for $c = C(MU)$. Thus for, say, $k = 60$, the adversary has no effective control over the congruence (4) that P will solve.

4. INVERSION IS EQUIVALENT TO FACTORIZATION

We now want to show that if an adversary can invert $E_{n,b}(x)$ by any algorithm then he can factor n. By inverting we mean finding for y one of the four x such that $E_{n,b}(x) = y$. Finding one such x is sufficient for the would be signature forger, so that we want to show that this is hard. Thus the problem of, say, forging P's signatures is exactly as intractable as the factorization of a number n which is a product of large primes. As mentioned in the Introduction, the scheme in [6] is *at most* as safe as factorization but conceivably easier to crack.

In the following theorem we count an addition of numbers $a,b, \leq n$ as one operation.

It is readily seen that if we can solve (4) for fixed n,b and arbitrary c then we can extract square roots, i.e., solve $y^2 \equiv m \bmod n$ whenever a solution exists. Namely, letting $b \equiv 2d \bmod n$ (n is odd) and $m = c + d^2 \bmod n$, (4) turns into

$$x^2 + 2dx + d^2 = (x+d)^2 \equiv m \bmod n.$$

Thus our result follows from

<u>THEOREM 1</u>: Let AL be an algorithm for finding one of the solutions of

$$(5) \qquad y^2 \equiv m \bmod n$$

whenever a solution exists, and requiring $F(n)$ steps. There exists an algorithm for factoring n requiring $2F(n) + 2\log_2 n$ steps.

Proof. Assume that $n = p \cdot q$ is a product of two primes, the case relevant for $E_{n,b}$. The proof easily extends to the general case.

For any $0 < k < n$, $(k,n) = 1$, there are exactly four solutions for the congruence

$$y^2 \equiv k^2 \bmod n.$$

Namely, let $res(k,p) = r$, $res(k,q) = s$ then the solutions y of this congruence satisfy $res(y,p) \equiv \pm r \bmod p, res(y,q) = \pm s \bmod q$ and each of the four sign combinations gives rise to a different solution. Defining for $0 \leq y_1, y_2 < n, y_1 \sim y_2$ to mean $y_1^2 \equiv y_2^2 \bmod n$, we see that this equivalence relation decomposes the set $0 < y < n$, $(y,n) = 1$ into classes each containing four elements.

Denote by $\sqrt{m}$ the solution of (5) by AL for any m, $(m,n) = 1$. If AL produces more than one solution then

the factorization algorithm that follows is even further facilitated.

Choose at random a number $0 < k < n$. If $(k,n) \neq 1$ then we directly get a factor of n. In practice, this possibility can be neglected. Compute $k^2 \equiv m \bmod n$.

Compute $k_1 = \sqrt{m}$ by AL. Now, k is in the equivalence class, by the relation $\sim$, of k_1. In a random choice of $0 < k < n$, all four possible choices of numbers within any class are equally likely. Hence with probability 1/2

$$k \equiv k_1 \bmod p,\ k \equiv -k_1 \bmod q$$

or

$$k \equiv -k_1 \bmod p,\ k \equiv k_1 \bmod q$$

Therefore with probability 1/2

(6) $$(k-k_1, n) = p \text{ or } q.$$

The computation of $\sqrt{m}$ requires $F(n)$ steps. The computation of the g.c.d. (6) requires at most $\log_2 n$ subtractions and divisions by 2, of numbers smaller than n. Hence the expected number of steps is $2F(n) + 2\log_2 n$.

If we count bit-operations then subtraction of numbers smaller than n requires at most $\log_2 n$ bit-operations and the bound is $2F(n) + 2(\log_2 n)^2$.

The previous theorem may be strengthened to cover the situation that for the given key $E_{n,b}$ can be decoded in just a small percentage of all cases.

<u>THEOREM 2</u>: If AL solves (5) in $F(n)$ steps for $1/e$ of the $0 < m < n$, $(m,n) = 1$, for which (5) has a solution, then there is an algorithm for factoring n requiring $2eF(n) + 2\log_2 n$ steps.

Proof. As in the proof of Theorem 1, choose a $0 < k < n$ at random and compute $k^2 \equiv m \bmod n$. Apply AL to find $\sqrt{m}$. If the computation runs more than $F(n)$ steps abort it and choose another k. Whenever a root $k_1 = \sqrt{m}$ is found, compute $(k-k_1,n)$. The analysis in the proof of Theorem 1 implies that with probability 1/2 each such try produces a factorization of n.

The expected number of choices of k leading to a $\sqrt{m}$ is e and the expected number of *successes* of AL needed for a factorization, is 2. Thus the total expected number of steps is $2eF(n) + 2\log_2 n$. Note that we embark on the second phase of the factorization only after a success of AL in finding $\sqrt{m}$.

If for example $e = 1000$, and $F(n)$ were not prohibitively large, then an adversary could factor n in $2000\ F(n) + 2\log_2 n$ steps. Consequently, if no practical algorithm for factoring n is possible, then no practical decoding algorithm could work in even 1/1000 of all cases.

5. GENERALIZATIONS

The above method of construction of a one-way function can be extended to employ polynomials or powers of x of small degrees other than 2.

Assume for example that $n = p \cdot q$, where p and q are primes of the form $3k + 1$. The one-way function will be $E(x) \equiv x^3 \bmod n$. The decoding is effected by solving $x^3 - m \equiv 0 \bmod p$ and $\bmod q$ by a probabilistic algorithm similar to the one used in Section 2. Again one can prove that any algorithm for extracting cubic roots leads, for n of the above form, to a factorization of n.

The probability that $x^3 \equiv w \bmod n$ is solvable for a random w is 1/9. Thus for utilization in signatures the quadratic scheme seems best.

REFERENCES

[1] Berlekamp, E.R., Factoring polynomials over large finite fields, Math. of Comp., vol. 24 (1970), pp. 713-735.

[2] Diffie, W., and Hellman, M., New directions in cryptography, IEEE Trans. of Inf. Theory, IT-22, (November 1976), pp. 644-654.

[3] Rabin, M., O., Probabilistic algorithms, Algorithms and Complexity, Recent Results and New Directions, J.F. Traub, editor, Academic Press, New York, 1976, pp. 21-40.

[4] Rabin, M.O., Digitalized signatures, Foundations of Secure Computations, R. Lipton and R. DeMillo editors, Academic Press New York, 1978.

[5] Rabin, M.O., Probabilistic algorithms in finite fields, submitted for publication.

[6] Rivest, R.L., Shamir A., and Adleman L., A method for obtaining digital signatures and public-key cryptosystems, Comm. ACM. vol. 21 (February 1978).

[7] Solovay, R., and Strassen, V., A fast monte-carlo test for primality, SIAM Jour. on Comp. Vol. 6 (1977), pp. 84-85.

R. Lidl

Express Mail No. EL636502710

CG

esidues, we have to use
ile we don't know if the
't is still an open problem
nputing logarithms over
a step forward towards
m.

phy", *IEEE Transactions*

tems, Ph.D. Dissertation,
ne 1979.

:rete Logarithm Problem
the 20th Annual FOCS

' Discrete Logarithms in
ara, CA *August 1982.*

ley.

3.

e Numbers". *Journal of*

. *Math. Comp. vol. 36 no.*

ison Wesley.

ison Wesley.

raphy. Ph.D. Dissertation,
une 1977.

ach". *Math. of comp. vol.*

itive roots. Royal society
968.

PERMUTATION POLYNOMIALS IN RSA-CRYPTOSYSTEMS

Rudolf Lidl and Winfried B. Müller

Department of Mathematics
University of Tasmania
Hobart, Australia 7001

Institut für Mathematik
Universität Klagenfurt
A-9020 Klagenfurt, Austria

1. INTRODUCTION

For the transmission of information in public-key cryptosystems a receiver R makes an enciphering key E_R public and keeps a deciphering key D_R secret. A sender S can read R's public key E_R and enciphers a message M as $E_R(M)$. Only the authorised receiver R knows the correct key D_R to reproduce the massage M by forming $D_R(E_R(M)) = M$. Here the key E_R has to be computationally easy to handle, but it has to be computationally infeasible to derive D_R from the knowledge of E_R above. If sender S wants to "sign" the message, she sends $E_R(D_S(M))$ and the receiver deciphers it as $E_S(D_R(E_R(D_S(M)))) = M$. Here E_S and D_S are the enciphering and deciphering keys, respectively, of S.

In order to have signatures like this we have to have the property $E_X \circ D_X = D_X \circ E_X$ for the keys of a person X. Particularly simple to handle are key functions with the properties

$$(1.1) \quad E_X \circ E_Y = E_Y \circ E_X, \; E_X \circ D_Y = D_Y \circ E_X, \; D_X \circ D_Y = D_Y \circ D_X$$

for any persons X and Y. If (1.1) holds we do not have to be concerned about the order of the composition of key-functions.

In the RSA-cryptosystem the key E_R can be regarded as a permutation of a set A of elements (numbers) used for enciphering a plain text, D_R is the inverse permutation to E_R on A. Permutations of the set $A = Z_m$ of residue classes modulo m can be obtained by using permutation polynomials modulo m.

These are polynomials which induce a permutation of Z_m on substitution of the elements of Z_m. In the RSA system the permutation polynomials x^k of Z_m are used, where $(k,\phi(m)) = 1$, $m = pq$ and p,q are (large) primes. These permutation polynomials form a group with respect to composition $\circ$, we have $x^k \circ x^l = x^{kl}$, $k,l \geq 1$. In general it is difficult to construct permutation polynomials whose inverses are known or are not too complicated to construct.

In this paper we study some questions connected with the RSA-cryptosystem and its generalisations. We investigate classes of polynomials and rational functions for which the task of finding inverses is easy and which are suitable for RSA-type cryptosystems.

2. POLYNOMIALS IN ONE VARIABLE

In the RSA-cryptosystem the polynomials x^k are used for enciphering modulo m. The polynomial functions $x \to x^k$ from Z_m into itself satisfy conditions (1.1). Müller and Nöbauer [11] suggested to replace the polynomials x^k by the Dickson polynomials $g_k(a,x)$ to create a modified RSA-cryptosystem. These polynomials are defined by

$$g_k(a,x) = \sum_{i=0}^{k/2} \frac{k}{k-i}\binom{k-i}{i}(-a)^i x^{k-2i}, \text{ for } a = \pm 1 .$$

For $a = 0$ we obtain x^k. The polynomials $g_k(a,x)$ also satisfy (1.1), since $g_k(a,x) \circ g_n(a,x) = g_{kn}(a,x)$. For $a = 1$ there is a simple recurrence relation, cf. [3], for generating these polynomials

$$g_{k+2} - x g_{k+1} + g_k = 0,\ g_0 = 2,\ g_1 = x .$$

If $m = pq$, p and q prime, then $g_k(a,x)$ induces a permutation of Z_m if and only if $(k,(p^2-1)(q^2-1)) = 1$ (cf. [4], [13]). In [4] it is also shown that $g_n(a,x)$ is the inverse of $g_k(a,x)$ if and only if $kn \equiv 1 \pmod{(p^2-1)(q^2-1)}$. It is impossible to calculate n, and therefore the inverse of $g_k(a,x)$, if the prime factors p and q of m are unknown.

In this section we investigate which other classes of polynomials in one variable can be used for modified forms of the RSA-cryptosystem. We suppose that any such class should satisfy (1.1). Since we want to have a cryptosystem in Z_m for an arbitrary product $m = pq$ of primes, we require that the desired classes of polynomials contain at least one of degree k for any positive integer k.

Following Lausc
over Q is a permutab
degree > 0, for k >
and f(x) and g(x) co
In order to find cla
with the above prope
Z. A chain C_1 = {l
(over Q) of a permut
is a linear polynomi
an equivalence relat
Theorem 3.33 of [5]
some congugate of ei
the chain of Chebysh
polynomial of the fi
over Q contains exac

All permutable
those chains which c
permutable chains ov
over Z. There is a
polynomials and the
namely

$g_k(a,x) = 2(\sqrt{a})$

Thus the polynomial
cryptosystems, form
[5] states that all
conjugates of S and
polynomials for an
that class replaces
a generalization.

Theorem 2.1 All p
RSA-type cryptosys

$\left(\frac{x}{u} - \frac{v}{u}\right) \circ S \circ$

and

$\left(\frac{x}{u} - \frac{v}{u}\right) \circ T \circ$

where $S = \{x, x^2, x$
polynomial of the

In summary,
and the Dickson (
essentially the o
polynomial of deg
polynomials of a

Following Lausch and Nöbauer [5] a class C of polynomials over Q is a permutable chain if every polynomial in C is of degree > 0, for k > 0 there exists a polynomial in C of degree k and f(x) and g(x) commute, that is $f \circ g = g \circ f$ for all $f, g \in C$. In order to find classes of polynomials for RSA-type cryptosystems with the above properties, we have to find permutable chains over Z. A chain $C_1 = \{l^{-1} \circ f_i \circ l \mid i \in I\}$ is called a conjugate (over Q) of a permutable chain $C = \{f_i \mid f_i \in Q[x], i \in I\}$, where l is a linear polynomial. C_1 is permutable too and conjugacy is an equivalence relation on the set of all permutable chains over Q. Theorem 3.33 of [5] proves that every permutable chain over Q is some congugate of either the chain of powers $S = \{x, x^2, x^3, \ldots\}$ or the chain of Chebyshev polynomials $T = \{t_i \mid t_i$ the ith Chebyshev polynomial of the first kind$\}$. Therefore any permutable chain over Q contains exactly one polynomial of degree k.

All permutable chains over Z are obtained by determining those chains which consist of polynomials over Z amongst the permutable chains over Q. The chains S and T are permutable over Z. There is a simple connection between the Dickson polynomials and the Chebyshev polynomials of the first kind, namely

$$g_k(a,x) = 2(\sqrt{a})^k \, t_k\left(\frac{x}{2\sqrt{a}}\right) .$$

Thus the polynomials x^k and $g_k(a,x)$, which can be used in RSA-cryptosystems, form permutable chains. Now proposition 3.51 of [5] states that all permutable chains over Z are certain conjugates of S and T. This shows us how to find all classes of polynomials for an RSA-type cryptosystem, where a polynomial of that class replaces x^k in the standard RSA-cryptosystem to lead to a generalization.

<u>Theorem 2.1</u> All possible classes of commuting polynomials for an RSA-type cryptosystem are given by the permutable chains

$$\left(\frac{x}{u} - \frac{v}{u}\right) \circ S \circ (ux+v) \ , \ u,v \in Z, \ u \neq 0, \ v^2-v \in Zu$$

and

$$\left(\frac{x}{u} - \frac{v}{u}\right) \circ T \circ (ux+v) \ , \ u,v \in Z, \ u \neq 0, \ v-2 \in Zu,$$

where $S = \{x, x^2, x^3, \ldots\}$ and $T = \{t_i \mid t_i$ the ith Chebyshev polynomial of the first kind$\}$.

In summary, theorem 2.1 shows that the power polynomials x^k and the Dickson (or Chebyshev) polynomials of degree k are essentially the only classes of polynomials such that there is a polynomial of degree k in the class for any $k \in N$ and that the polynomials of a class commute.

3. ON THE CHOICE OF KEYS IN THE RSA-CRYPTOSYSTEM

In the original RSA-cryptosystem it is important to choose the enciphering function P_k $x \to x^k$ from Z_{pq} into itself in such a way that P_k has as few fixed points as possible. This problem has been considered widely, see e.g. [2], [12], [15]. A result contained, for instance, in [12] says that the permutation P_k on Z_{pq} has exactly $((k-1,p-1)+1)((k-1,q-1)+1)$ fixed points. P_k is a permutation of Z_{pq} if and only if $(k,(p-1)(q-1)) = 1$. Moreover, for $k = (p-2)(q-2)$ the permutation P_k of Z_{pq} has exactly 9 fixed points, since

$$(p-2)(q-2) = ((\tfrac{p-1}{2}-1)2+1)((\tfrac{q-1}{2}-1)2+1) .$$

This shows

<u>Proposition 3.1</u> For odd primes p and q the number of fixed points of a permutation P_k of Z_{pq} is always an odd integer ≥ 9. For $k = (p-2)(q-2)$ P_k has exactly 9 fixed points.

Let $[a,b]$ denote the least common multiple of integers a and b. All k with $(k,(p-1)(q-1)) = 1$, $(k-1,p-1) = d_1$ and $(k-1,q-1) = d_2$ are obtained by letting $k = rd_1d_2 + 1$, where r is any integer satisfying $(r,(p-1)(q-1)/d_1d_2) = 1$.

<u>Theorem 3.2</u> Let $d_1 = (k-1,p-1)$, $d_2 = (k-1,q-1)$. All permutations P_k of Z_{pq} with exactly $(d_1+1)(d_2+1)$ fixed points are obtained, if we set $k = rd_1d_2 + 1$ and r is any of the integers $1,\ldots,[p-1,q-1]$, which satisfies $(r,(p-1)(q-1)/d_1d_2) = 1$.

Now let $[p-1,q-1] = q_1^{e_1}\cdots q_s^{e_s}$ and $[d_1,d_2] = q_1^{f_1}\cdots q_s^{f_s}$ be decompositions into prime factors, then [12] shows that the number of k's in theorem 3.2 is given by

$$N = \prod_{j=1}^{s} q_j^{e_j-f_j-1}(q_j-t_j) \text{ where } t_j = \begin{cases} 2 & \text{if } f_j = 0 \\ 1 & \text{if } 0 < f_j < e_j \\ 0 & \text{if } f_j = e_j \end{cases} .$$

This result implies

<u>Proposition 3.3</u> For odd primes p and q the number of permutations P_k with exactly 9 fixed points is

$$2^{e_1-2}\prod_{j=2}^{s} q_j^{e_j-1}(q_j-2) .$$

4. POLYNOMIALS IN S

A possible gener
achieved by consideri
x^k or $g_k(a,x)$. Such
Instead of permutatio
polynomial vectors, o
Niederreiter [8, ch.7

$(p_1(x_1,\ldots,x_n),\ldots$

is called an orthogon
permutation of Z_m^n on
$(x_1,\ldots,x_n)$.

To give a direct
n-tuples of the small
used as the code alph
message into n-tuples
orthogonal system fo
for $i=1,\ldots,n$. For
satisfy $k_il_i \equiv 1$ (mod
Then $(x_1^{k_1l_1},\ldots,x_n^{k_n}$

Corresponding to
set of polynomials i
for Z_m, namely the D
variables. These p
their algebraic, ana
[3], [8], [9], [10].
and $m = pq$, but all
integers m.

We define the D

$g_k(x,y) = u^k +$

where $x = u + v + u$
are elements of C.
in x and y with inte
recursive relations

$g_{k+3}(x,y) - xg$

with initial condit

$\bar{g}_{k+3}(x,y) - y\bar{g}$

4. POLYNOMIALS IN SEVERAL VARIABLES

A possible generalization of the RSA-cryptosystem can be achieved by considering polynomials in several variables instead of x^k or $g_k(a,x)$. Such a generalization has been suggested in [11]. Instead of permutation polynomials one has to study permutation polynomial vectors, or orthogonal systems (cf. Lidl and Niederreiter [8, ch.7]). Let m be as above. A vector

$$(p_1(x_1,\dots,x_n),\dots,p_n(x_1,\dots,x_n)) \in (Z[x_1,\dots,x_n])^n$$

is called an orthogonal system for Z_m^n, if this vector induces a permutation of Z_m^n on substitution of $(a_1,\dots,a_n) \in Z_m^n$ for $(x_1,\dots,x_n)$.

To give a direct generalization of the RSA-cryptosystem the n-tuples of the smallest non-negative respresentative of Z_m can be used as the code alphabet, or alternatively, one subdivides the message into n-tuples for enciphering. For example, a simple orthogonal system for Z_m is $(x_1^{k_1},\dots,x_n^{k_n})$, where $(k_i,\phi(m)) = 1$ for $i=1,\dots,n$. For deciphering one has to find l_i's which satisfy $k_i l_i \equiv 1 \pmod{\phi(m)}$ and then forms $(x_1^{l_1},\dots x_n^{l_n})$. Then $(x_1^{k_1},\dots,x_n^{k_n}) \circ (x_1^{l_1},\dots,x_n^{l_n}) = (x_1,\dots,x_n)$.

Corresponding to the Dickson polynomials $g_k(a,x)$ there is a set of polynomials in n variables which form an orthogonal system for Z_m, namely the Dickson (or Chebyshev) polynomials in n variables. These polynomials have been studied extensively as to their algebraic, analytic and number-theoretic properties, see [3], [8], [9], [10]. Here we only describe the simple case $n = 2$ and $m = pq$, but all results hold for arbitrary n and squarefree integers m.

We define the Dickson polynomials according to [8] as

$$g_k(x,y) = u^k + v^k + u^{-k}v^{-k}, \quad \bar{g}_k(x,y) = u^{-k} + v^{-k} + u^k v^k$$

where $x = u + v + u^{-1}v^{-1}$ and $y = u^{-1} + v^{-1} + uv$; here u and v are elements of C. It can be verified that g and $\bar{g}$ are polynomials in x and y with integral coefficients. These polynomials satisfy recursive relations of the form

$$g_{k+3}(x,y) - x g_{k+2}(x,y) + y g_{k+1}(x,y) - g_k(x,y) = 0$$

with initial conditions $g_0 = 3$, $g_1 = x$, $g_2 = x^2 - 2y$,

$$\bar{g}_{k+3}(x,y) - y\bar{g}_{k+2}(x,y) + x\bar{g}_{k+1}(x,y) - \bar{g}_k(x,y) = 0$$

with initial conditions $\bar{g}_0 = 3$, $\bar{g}_1 = y$, $\bar{g}_2 = y^2 - 2x$. An explicit expression of these polynomials is given in [8, chapter 7]:

$$g_k(x,y) = \sum_{\substack{i=0}}^{k/2} \sum_{\substack{j=0 \\ 2i+3j \le k}}^{k/3} \frac{k(-1)^i}{k-i-2j} \binom{k-i-2j}{i+j} \binom{i+j}{i} x^{k-2i-3j} y^i$$

and

$$\bar{g}_k(x,y) = g_k(y,x) .$$

In [9] it is shown that $(g_k, \bar{g}_k)$ form an orthogonal system for Z_p^2 iff $(k, p^s - 1) = 1$ for $s = 1,2,3$. For $m = pq$ as above, Matthews [10] proved that $(g_k(x,y), \bar{g}_k(x,y))$ form an orthogonal system for Z_m^2 iff $(k,L) = 1$, where

$$L = \mathrm{lcm}\{\mathrm{lcm}\{p-1, p+1, p^2+p+1\}, \mathrm{lcm}\{q-1, q+1, q^2+q+1\}\} . \tag{4.1}$$

The definition of the polynomials in terms of a functional equation implies

$$(g_k, \bar{g}_k) \circ (g_l, \bar{g}_l) = (g_{kl}, \bar{g}_{kl}) . \tag{4.2}$$

Clearly $(g_1, \bar{g}_1) = (x,y)$. Also $(g_k, \bar{g}_k) = (g_1, \bar{g}_1)$ on Z_m^2 iff

$$k \equiv 1 \pmod{L} . \tag{4.3}$$

This shows how to find the inverse pair to a given pair of polynomials, namely by solving

$$kl \equiv 1 \pmod{L} . \tag{4.4}$$

In summary, to use $(g_k, \bar{g}_k)$ for an RSA-type cryptosystem, we subdivide the message into pairs (a,b) of integers $< m$, encipher them as $(g_k(a,b), \bar{g}_k(a,b))$ modulo m and transmit this pair of integers (mod m) to the receiver. For deciphering, the receiver finds an l satisfying (4.4) and forms the composite (4.2), which recovers (a,b). Since L is impossible to calculate without knowing the factors of m, this procedure appears to be secure for large primes p and q. As before, m and $(g_k, \bar{g}_k)$ are the public key. Only the authorised receiver knows the prime factors and can thus calculate the inverse vector $(g_l, \bar{g}_l)$.

5. ON PERMUTATION FUNCTIONS

In this section we consider the problem of replacing polynomials, such as x^k or $g_k(a,x)$, in the RSA-cryptosystem by rational functions that induce permutations (mod m).

Levine and Brawley [
rational functions o

Let $r(x) = g(x)$
where g and h are re
permutation function
is a prime residue c
$Z_m \to Z_m$, $\pi(b) \equiv h(b)$
A polynomial $g(x)$ is
permutation function
is a permutation fun
(mod p) and (mod q).
also [7]). Sometim
It is assumed that
for $b \neq 0$. If the
then $r(x)$ is called
A rational function
and for Z_m iff the d
the degree of the de
functions of this ty
to Z_m.

Rédei [14] stud
induced by certain
also be used for cry
and let $(\frac{\alpha}{p}) = -1$ and

$(x + \sqrt{\alpha})^n = g_n$

where $g_n(x)$ and $h_n($

$g_n(x) = \sum_{i=0}^{n/2} \binom{n}{2i}$

Rédei defines a rat
that $f_n(x)$ is a per
odd and $(n,p+1) = 1$

(5.1) $\left(\frac{x + \sqrt{\alpha}}{x - \sqrt{\alpha}}\right)^n$

This yields

$\frac{f_{kn}(x) +}{f_{kn}(x) -}$

Levine and Brawley [6] give a simple example of the use of linear rational functions over finite fields.

Let $r(x) = g(x)/h(x)$ be a quotient of polynomials over Z, where g and h are relatively prime in $Z[x]$. $r(x)$ is called a permutation function modulo a positive integer m if $h(b)$ (mod m) is a prime residue class (mod m) for any $b \in Z$ and the mapping $\pi : Z_m \to Z_m$, $\pi(b) \equiv h(b)^{-1}g(b)$ (mod m) is a permutation. A polynomial $g(x)$ is a permutation polynomial iff $g(x)/1$ is a permutation function. If $m = pq$, for primes p and q, then $r(x)$ is a permutation function (mod m) iff it is a permutation function (mod p) and (mod q). (Nöbauer [13] studied the case $m = p^e$, see also [7]). Sometimes it is convenient to adjoin a symbol ∞ to Z_m. It is assumed that $\infty = 1/0, 0 = 1/\infty$, $\infty + b = \infty$ for $b \in Z_m$, $b\infty = \infty$ for $b \neq 0$. If the quantities $r(b)$ are distinct for all $b \in Z_m \cup \{\infty\}$ then $r(x)$ is called a permutation function over $Z_m \cup \{\infty\}$. A rational function $r(x)$ is a permutation function for $Z_m \cup \{\infty\}$ and for Z_m iff the degree of the numerator of $r(x)$ is greater than the degree of the denominator. We shall only consider rational functions of this type and therefore restrict our considerations to Z_m.

Rédei [14] studied permutations of finite fields which are induced by certain rational functions $r_n(x)$. These functions can also be used for cryptosystems. Let $\alpha \neq 0$ be a nonsquare integer and let $(\frac{\alpha}{p}) = -1$ and $(\frac{\alpha}{q}) = -1$. We set

$$(x + \sqrt{\alpha})^n = g_n(x) + h_n(x)\sqrt{\alpha}$$

where $g_n(x)$ and $h_n(x)$ are polynomials over Z, given explicitly as

$$g_n(x) = \sum_{i=0}^{n/2} \binom{n}{2i} \alpha^i x^{n-2i} \; , \quad h_n(x) = \sum_{i=0}^{n/2} \binom{n}{2i+1} \alpha^i x^{n-2i-1} \; .$$

Rédei defines a rational function $f_n(x) = g_n(x)/h_n(x)$ and proves that $f_n(x)$ is a permutation function modulo a prime $p \neq 2$, if n is odd and $(n,p+1) = 1$, $p \nmid n$. The construction of $f_n(x)$ is such that

$$\left(\frac{x + \sqrt{\alpha}}{x - \sqrt{\alpha}}\right)^n = \frac{f_n(x) + \sqrt{\alpha}}{f_n(x) - \sqrt{\alpha}} \; . \tag{5.1}$$

This yields

$$\frac{f_{kn}(x) + \sqrt{\alpha}}{f_{kn}(x) - \sqrt{\alpha}} = \frac{f_k(f_n(x)) + \sqrt{\alpha}}{f_k(f_n(x)) - \sqrt{\alpha}} \; .$$

Therefore

(5.2) $f_k(f_n(x)) = f_{kn}(x)$,

which is an essential property for a function to be useful for our purpose. Let π_n be the permutation of Z_p induced by $f_n(x)$, then the product rule $\pi_k\pi_n = \pi_{kn}$ corresponds to (5.2), that is the product of two permutations of Z_p is induced by the composite of the polynomials belonging to the factors. Moreover, $\pi_k = \pi_n$ iff $k \equiv n \pmod{p+1}$ (cf. Rédis [14]). We note that $f_1(x) = x$ and $\pi_1 = \varepsilon$ (the identity map) iff $(p+1)|(n-1)$. Therefore we can easily find the inverse of a given $f_n(x)$ on Z_p according to

<u>Lemma</u> $f_k(f_n(x)) = f_n(f_k(x)) = f_1(x) = x$ iff

(5.3) $nk \equiv 1 \pmod{p+1}$.

The proof is essentially contained in [14].

Now we can state the use of $f_n(x)$ in RSA-cryptosystems. Let $m = pq$, p,q two large primes which are kept secret, let n be an odd integer with $n \nmid p$, $n \nmid q$, $(n,p+1) = (n,q+1) = 1$. α is as given above. Then $f_n(x)$ is a permutation function modulo m. This follows immediately from the Chinese remainder theorem. A message is encoded as an integer $a < m$ and then enciphered as $f_n(a) \pmod m$. The receiver deciphers this cipher by calculating the inverse $f_k(x)$ of $f_n(x) \pmod m$. The receiver has to find a k satisfying (5.3) (mod $p+1$) and (mod $q+1$), or equivalently

(5.4) $nk \equiv 1 \pmod{[p+1,q+1]}$.

Again, without knowing the factors of m it is impracticable to find a k satisfying (5.4) and with it the inverse of $f_n(x)$. In [13] it is shown that there are infinitely many primes p and q with $(p+1,n) = (q+1,n) = 1$ and $(\frac{\alpha}{p}) = (\frac{\alpha}{q}) = -1$, n odd and α a nonsquare, except in the case when the squarefree kernel of α equals -3 and at the same time $3|n$.

REFERENCES

[1] Carlitz, L.: A note on permutation functions over a finite field. Duke Math. J. 29, 325-332 (1962)

[2] Ecker, A.: Finite semigroups and the RSA-cryptosystem. Lecture Notes in Computer Science, vol.149, Springer-Verlag, 353-370 (1983).

[3] Eier, R. and R. Lidl: Tschebyscheffpolynome in einer und zwei Variablen. Abh. Math. Seminar Univ. Hamburg <u>41</u>, 17-27 (1974)

[4] Lausch, H., W. M
einer durch
gruppe des
Math. <u>261</u>,

[5] Lausch, H. and W
Holland, Am

[6] Levine, J. and J
of permutat

[7] Lidl, R.: Regul
Beiträge Al

[8] Lidl, R. and H.
of Mathemat
Wesley, Rea

[9] Lidl, R. and Ch.
variables.

[10] Matthews, R.:
induced by
integers m
88-103 (19.

[11] Müller, W.B. an
cryptosyst
(to appear

[12] Müller, W.B. an
Potenzperm
Naturwiss.

[13] Nöbauer, W.: U
funktionen
230-238 (1

[14] Rédei, L.: Übe
endlichen
85-92 (194

[15] Smith, D.R.:
Adleman cr

[4] Lausch, H., W. Müller and W. Nöbauer: Über die Struktur einer durch Dicksonpolynome dargestellten Permutations gruppe des Restklassenringes modulo n. J. Reine Angew. Math. 261, 88-99 (1973).

[5] Lausch, H. and W. Nöbauer: Algebra of Polynomials, North Holland, Amsterdam, 1973.

[6] Levine, J. and J.V. Brawley: Some cryptographic applications of permutation polynomials. Cryptologia 1, 76-92 (1977).

[7] Lidl, R.: Reguläre Polynome über endlichen Körpern. Beiträge Alg. Geometrie 2, 58-59 (1974).

[8] Lidl, R. and H. Niederreiter: Finite Fields. Encyclopedia of Mathematics and its Applications vol.20. Addison-Wesley, Reading, Massachusetts, 1983.

[9] Lidl, R. and Ch. Wells: Chebyshev polynomials in several variables. Journal reine angew. Math. 255, 104-111 (1972).

[10] Matthews, R.: The structure of the group of permutations induced by Chebyshev polynomial vectors over the ring of integers mod m. J. Austral. Math. Soc. Ser. A, 32, 88-103 (1982).

[11] Müller, W.B. and W. Nöbauer: Some remarks on public-key cryptosystems. Studia Sci. Math. Hungar. 16, (1981) (to appear).

[12] Müller, W.B. and W. Nöbauer: Über die Fixpunkte der Potenzpermutationen. Österr. Akad. Wiss. Math. Naturwiss.Kl. Sitzungsber.II (1983) (to appear).

[13] Nöbauer, W.: Über Permutationspolynome und Permutationsfunktionen für Primzahlpotenzen. Monatsh. Math. 69, 230-238 (1965).

[14] Rédei, L.: Über eindeutig umkehrbare Polynome in endlichen Körpern. Acta Sci. Math. (Szeged) 11, 85-92 (1946).

[15] Smith, D.R.: Universal fixed messages and the Rivest-Shamir-Adleman cryptosystem. Mathematika 26,44-52 (1979).

Generating a Product of Three Primes With an Unknown Factorization

Dan Boneh
dabo@cs.stanford.edu

Jeremy Horwitz
horwitz@cs.stanford.edu

Computer Science Department,
Stanford University,
Stanford, CA, 94305-9045

Abstract

We describe protocols for three or more parties to jointly generate a composite $N = pqr$ which is the product of three primes. After our protocols terminate N is publicly known, but neither party knows the factorization of N. Our protocols require the design of a new type of distributed primality test for testing that a given number is a product of three primes. We explain the cryptographic motivation and origin of this problem.

1 Introduction

In this paper, we describe how three (or more) parties can jointly generate an integer N which is the product of three prime number $N = pqr$. At the end of our protocol the product N is publicly known, but neither party knows the factorization of N. Our main contribution is a new type of probabilistic primality test that enables the three parties to jointly test that an integer N is the product of three primes without revealing the factorization of N. Our primality test simultaneously uses two groups: the group $\mathbb{Z}_N^*$ and the projective line over $\mathbb{Z}_N$.

The main motivation for this problem comes from cryptography, specifically the sharing of an RSA key. Consider classical RSA: $N = pq$ is a public modulus, e is a public exponent and d is secret where $de = 1 \bmod \varphi(N)$. At a high level a digital signature of a message M is obtained by computing $M^d \bmod N$. In some cases the secret key d is highly sensitive (e.g. the secret key of a Certification Authority) and it is desirable to avoid storing it at a single location. Splitting the key d into a number of pieces and storing each piece at a different location avoids this *single point of failure.* One approach (due to Frenkel [8]) is to pick three random numbers satisfying $d = d_1 + d_2 + d_3 \bmod \varphi(N)$ and store each of the shares d_1, d_2, d_3 at one of three different sites. To generate a signature of a message M site i computes $S_i = M^{d_i} \bmod N$ for $i = 1, 2, 3$ and sends the result to a *combiner.* The combiner multiplies the S_i and obtains the signature $S = S_1 S_2 S_3 = M^d \bmod N$. If one or two of the sites are broken into, no information about the private key is revealed. An important property of this scheme is that it produces standard RSA signatures – the user receiving the signature is totally unaware of the extra precautions taken in protecting the private key. Note that during signature generation the secret key is never reconstructed at a single location.

To provide fault tolerance one slightly modifies the above technique to enable any two of the three sites to generate a signature. This way if one of the sites is temporarily unavailable the Certification

Authority can still generate signatures using the remaining two sites. If the key was only distributed among two sites the system would be highly vulnerable to faults.

We point out that classic techniques of secret sharing [15] are inadequate in this scenario. Secret sharing requires one to reconstruct the secret at a single location before it can be used. hence introducing a single point of failure. The technique described above of sharing the secret key such that it can be used without reconstruction at a single location is known as *Threshold Cryptography*. See [10] for a succinct survey of these ideas and nontrivial problems associated with them.

An important question left out of the above discussion is key generation. Who generates the RSA modulus N and the shares d_1, d_2, d_3? Previously the answer was a *trusted dealer* would generate N and distribute the shares d_1, d_2, d_3 to the three sites. Clearly this solution is undesirable since it introduces a new single point of failure – the trusted dealer. It knows the factorization of N and the secret key d. If it is compromised the secret key is revealed. Recently Boneh and Franklin [2] designed a protocol that enables three (or more) parties to jointly generate an RSA modulus $N = pq$ and shares d_1, d_2, d_3 of a private key. At the end of the protocol the parties are assured that N is indeed the product of two large primes however non of them know its factorization. In addition each party learns exactly one of d_1, d_2, d_3 and has no computational information about the other shares. Thus, there is no need for a trusted dealer. We note that Cocks [6] introduced a heuristic protocol enabling two parties to generate a shared RSA key.

In this paper we design an efficient protocol enabling three (or more) parties to generate a modulus $N = pqr$ such that neither party knows the factorization of N. Once N is generated the same techniques used in [2] can be used to generate shares d_1, d_2, d_3 of a private exponent. For this reason throughout the paper we focus on the generation of the modulus $N = pqr$ and ignore the generation of the private key. The methods of [2] do not generalize to generate a modulus with three prime factors and new techniques had to be developed for this purpose.

We remark that techniques of *secure circuit evaluation* [1, 5, 17] can also be used to solve this problem. However, these protocols are mostly theoretical resulting in extremely inefficient algorithms.

2 Motivation

The problem discussed in the paper is a natural one and thus our solution is of independent interest. Nonetheless, the problem is well motivated by a method for improving the efficiency of shared generation of RSA keys. To understand this we must briefly recall the method used by Boneh and Franklin [2]. We refer to the three parties involved as Alice, Bob and Carol. At a high level to generate a modulus $N = pq$ the protocol works as follows:

Step 1 Alice picks two random n bit integers p_a, q_a, Bob picks two random n bit integers p_b, q_b and Carol picks two random n bit integers p_c, q_c. They keep these values secret.

Step 2 Using a private distributed computation they compute the value

$$N = (p_a + p_b + p_c)(q_a + q_b + q_c)$$

At the end of the computation N is publicly available however no other information about the private shares is revealed. This last statement is provable in an information theoretic sense.

Step 3 The three parties perform a distributed primality test to test that N is the product of exactly two primes. As before, this step provably reveals no information about the private shares.

Step (3), the distributed primality test, is a new type of probabilistic primality test which is one of the main contributions of [2]. Step (2) is achieved using an efficient variation of the BGW [1] protocol.

A drawback of the above approach is that both factors of N are simultaneously tested for primality. Hence, the expected number of times step (3) is executed is $O(n^2)$. This is much worse than single user generation of N where the two primes are first generated separately by testing $O(n)$ candidates and then multiplied together. When generating a 1024 bit modulus this results in significant slowdown when compared with single user generation.

To combat this quadratic slowdown one may try the following alternate approach.

Step 1 Alice picks a random n bit prime p and a random n bit integer r_a. Bob picks a random n bit prime q and a random n bit integer r_b. Carol picks a random n bit integer r_c. They keep these values secret.

Step 2 Using a private distributed computation they compute the value

$$N = pq(r_a + r_b + r_c)$$

At the end of the computation N is publicly available however no other information about the private shares is revealed.

Step 3 The three parties use the results of this paper to test that N is the product of exactly three primes. This step provably reveals no information about the private shares.

At the end of the protocol neither party knows the full factorization of N. In addition, this approach does not suffer from the quadratic slowdown observed in the previous method. Consequently, it is faster by roughly a factor of 50 (after taking effects of trial division into account). As before, step (2) is carried out by an efficient variant of the BGW protocol.

Instead of solving the specific problem of testing that $N = pq(r_a + r_b + r_c)$ is a product of three primes we solve the more general problem of testing that

$$N = (p_a + p_b + p_c)(q_a + q_b + q_c)(r_a + r_b + r_c)$$

is a product of three primes without revealing any information about the private shares. This primality test is the main topic of this paper.

For the sake of completeness we point out that in standard single party cryptography there are several advantages to using an RSA modulus $N = pqr$ rather than the usual $N = pq$ (the size of the modulus is the same in both cases). First, signature generation is much faster using the Chinese Remainder Theorem (CRT). When computing $M^d \bmod N$ one only computes $M^{d \bmod p-1} \bmod p$ for all three factors. Since the numbers (and exponents) are smaller signature generation is about twice as fast as using CRT with $N = pq$. Another advantage is that an attack on RSA due to Wiener [16] becomes less effective when using $N = pqr$. Wiener showed that for $N = pq$ if $d < N^{1/4}$ one can recover the secret key d from the public key. When $N = pqr$ the attack is reduced to $d < N^{1/6}$ and hence it may be possible to use smaller values of d as the secret key. Finally, we note that the fastest factoring methods [13] cannot take advantage of the fact that the factors of $N = pqr$ are smaller than those of a standard RSA modulus $N = pq$.

3 Preliminaries

In this section, we explain the initial setup for our new probabilistic primality test and how it is obtained. We then explain a basic protocol which we use in the later parts of the paper. At first reading the reader may wish to skip to Section 4 and take on faith that the necessary setup is attainable.

3.1 Communication and privacy model

The communication and privacy model assumed by our protocol are as follows:

Full connectivity Any party can communication with any other party. This is a typical setup on a local network or the Internet.

Private and authenticated channels Messages sent from party A to party B are private and cannot be tampered with en route. This simply states that A and B share a secret key which they can use for encryption and authentications.

Honest parties We assume all parties are honestly following the protocol. This is indeed the case when they are truly trying to create a shared key. This assumption is used by both [2] and [6]. We note that some recent work [9] makes the protocol of [2] robust against cheating adversaries at the cost of some slowdown in performance (roughly a factor of 100). These robustness results apply to the protocols described in this paper as well.

Collusion Our protocol is 1-private. That is to say that a single party learns no information about the factorization of $N = pqr$. However, if two of the three parties collude they can recover the factors. For three parties this is fine since our goal is to enable two-out-of-three signature generation. Hence, two parties are always jointly able to recover the secret key. More generally, when k parties participate in our primality test protocol one can achieve $\lfloor \frac{k-1}{2} \rfloor$ privacy. That is, any minority of parties learns no information about the factors of N.

3.2 Generations of N

In the previous section we explained that Alice, Bob and Carol generate N as

$$N = (p_a + p_b + p_c)(q_a + q_b + q_c)(r_a + r_b + r_c)$$

where party i knows p_i, q_i, r_i for $i = a, b, c$ and keeps these shares secret while making N publicly available. To compute N without revealing any other information about the private shares we use the BGW protocol [1]. For the particular function above the protocol is quite efficient requiring three rounds of communication and a total of 6 messages. The protocol is information theoretically secure, i.e. other than the value of N party i has no information about the shares held by other parties. This is to say the protocol is $1-$private.

We do not go into the details of how the BGW protocol is used to compute N since it is tangential to the topic of this paper — testing that N is a product of three distinct primes. For our purpose it suffices to assume N is public while the private shares are kept secret.

An important point is that our primality test can only be applied when $p_a + p_b + p_c = q_a + q_b + q_c = r_a + r_b + r_c = 3 \bmod 4$. Hence, the parties must coordinate the two lower bits of their shares ahead of time so that the sums are indeed 3 modulo 4. Indeed, this means that a priori each party knows the two least significant bits of the other's shares.

3.3 Sharing of $(p-1)(q-1)(r-1)$ and $(p+1)(q+1)(r+1)$

Let $p = p_a + p_b + p_c$, $q = q_a + q_b + q_c$ and $r = r_a + r_b + r_c$. We define $\hat{\varphi} = (p-1)(q-1)(r-1)$. Since p, q, r are not necessarily prime $\hat{\varphi}$ may not equal $\varphi(N)$. Our protocol requires that the value $\hat{\varphi}$ be shared additively among the three parties. That is, $\hat{\varphi} = \varphi_a + \varphi_b + \varphi_c$ where only party i knows φ_i for $i = a, b, c$.

An additive sharing of $\hat{\varphi}$ is achieved by observing that $\hat{\varphi} = N - pq - pr - qr + p + q + r - 1$. To share $\hat{\varphi}$ it suffices to represent $pq + pr + qr$ using an additive sharing $A + B + C$ among the three parties. The additive sharing of $\hat{\varphi}$ is then

$$\varphi_a = N - A + p_a + q_a + r_a - 1 \quad ; \quad \varphi_b = -B + p_b + q_b + r_b \quad ; \quad \varphi_c = -C + p_c + q_c + r_c$$

The conversion of $pq + pr + qr$ into an additive sharing $A + B + C$ is carried out using a simple variant of the BGW protocol used in the computation of N. The BGW protocol can be used to compute the value pq; however, instead of making the final result public the BGW variant shares the result additively among the three parties. The details of this variant can be found in [2, Section 6.2].

As before, we do not give the full details of the protocol for converting $pq + pr + qr$ into an additive sharing. Since we wish to focus on the primality test it suffices to assume that an additive sharing of $\hat{\varphi}$ is available in the form of $\varphi_a + \varphi_b + \varphi_c$.

In addition to a sharing of $\hat{\varphi}$ we also require an additive sharing of $\hat{\psi} = (p+1)(q+1)(r+1)$. Once an additive sharing of $pq + pr + qr$ is available it is trivial to generate an additive sharing of $\hat{\psi}$. Simply set

$$\psi_a = N + A + p_a + q_a + r_a + 1 \quad ; \quad \psi_b = B + p_b + q_b + r_b \quad ; \quad \psi_c = C + p_c + q_c + r_c$$

3.4 Comparison protocol

Our primality test makes use of what we call a *comparison protocol*. Let A be a value known to Alice, B a value known to Bob and C a value known to Carol. We may assume $A, B, C \in \mathbb{Z}_N^*$. The protocol enables the three parties to test that $ABC = 1 \bmod N$ without revealing any other information about the product ABC. We give the full details of the protocol in this section.

Let $P > N$ be some prime known to all parties. The protocol proceeds as follows:

Step 1. Carol picks a random element $C_1 \in \mathbb{Z}_N^*$ and sets $C_2 = CC_1^{-1} \bmod N$. Clearly $C = C_1C_2 \bmod N$. Carol then sends C_1 to Alice and C_2 to Bob.

Step 2. Alice sets $A' = AC_1$ and Bob sets $B' = (BC_2)^{-1} \bmod N$. Both values A' and B' can be viewed as integers in the range $[0, N)$. The problem is now reduced to testing whether $A' = B'$ (as integers) without revealing any other information about A and B.

Step 3. Alice picks a random $c \in \mathbb{Z}_P^*$ and $d \in \mathbb{Z}_P$. She sends c, d to Bob. Alice then computes $h(A') = cA' + d \bmod P$ and sends the result to Carol. Bob computes $h(B') = cB' + d \bmod P$ and sends the result to Carol.

Step 4. Carol tests if $h(A') = h(B') \bmod P$. If so, she announces that $ABC = 1 \bmod N$. Otherwise she announces $ABC \neq 1 \bmod N$.

The correctness and privacy of the protocol are stated in the next two lemmas. Correctness is elementary and is stated without proof.

Lemma 3.1 *Let $A, B, C \in Z_N^*$. At the end of the protocol the parties correctly determine if $ABC = 1 \bmod N$ or $ABC \neq 1 \bmod N$.*

Lemma 3.2 *The protocol is 1−private. That is, other than the result of the test each party learns no other information.*

Proof To prove the protocol is 1−private we provide a simulation argument for each party's view of the protocol. Alice's view of the protocol is made up of the values $A, C_1, c, d, h(A')$ and the final result of the test. These values can be easily simulated by picking C_1 at random in $\mathbb{Z}_N^*$, picking c at random in $\mathbb{Z}_P^*$ and d at random in $\mathbb{Z}_P$. This is a perfect simulation of Alice's view. A simulation argument for Bob is essentially the same.

Simulating Carol's view is more interesting. Carol's view consists of $C, C_1, C_2, h(A'), h(B')$ and the result of the test. The point is that $h(A')$ and $h(B')$ reveal no information about A and B since they are either equal, or random independent elements of $\mathbb{Z}_P$. Which of the two is determined by the result of the test. The independence follows since the family of hash functions $h(x) = cx + d \bmod P$ is a universal family of hash functions (i.e. not knowing c, d the values $h(x), h(y)$ are independent for any $x, y \in \mathbb{Z}_P$).

To simulate Carol's view the simulator picks $C_1, C_2 \in \mathbb{Z}_N^*$ at random so that $C = C_1C_2 \bmod N$. Then depending on the results of the test it either picks the same random element of $\mathbb{Z}_P$ twice or picks two random independent elements of $\mathbb{Z}_P$. This is a perfect simulation of Carol's view. This proves Carol gains no extra information from the protocol since given the outcome of the test, she can generate the values sent by Alice and Bob herself. □

4 The probabilistic primality test

We now describe the main primality test. As discussed in the previous section our primality test applies once the following setup is achieved:

Shares Each party i has three secret n-bit values p_i, q_i, r_i for $i = a, b, c$.

The modulus $N = (p_a + p_b + p_c)(q_a + q_b + q_c)(r_a + r_b + r_c)$ is public. We set $p = p_a + p_b + p_c$, $q = q_a + q_b + q_c$ and $r = r_a + r_b + r_c$. Throughout the section we are assuming that $p = q = r = 3 \bmod 4$. Thus, the parties must a priori coordinate the two least significant bits of their shares so that this condition holds.

Sharing $\hat{\varphi}, \hat{\psi}$: The parties share $(p-1)(q-1)(r-1)$ as $\varphi_a + \varphi_b + \varphi_c$ and $(p+1)(q+1)(r+1)$ as $\psi_a + \psi_b + \psi_c$.

Given this setup they wish to test that p, q and r are distinct primes without revealing p, q, r. At this point nothing is known about p, q, r other than $p = q = r = 3 \bmod 4$. Throughout the section we use the following notation:

$$\hat{\varphi} = \varphi_a + \varphi_b + \varphi_c = (p-1)(q-1)(r-1)$$
$$\hat{\psi} = \psi_a + \psi_b + \psi_c = (p+1)(q+1)(r+1)$$

Clearly if N is a product of three distinct primes then $\varphi(N) = \hat{\varphi}$. Otherwise, this equality may not hold.

Our primality test is made up of four steps. We first state what each step tests for and in the subsequent subsections explain how each step is carried out without revealing any information about the factors of N.

Step 1 The parties pick a random $g \in \mathbb{Z}_N^*$ and jointly test that $g^{\varphi_a+\varphi_b+\varphi_c} = 1 \bmod N$. If the test fails N is rejected. This step reveals no information other than the outcome of the test. We refer to this step as a Fermat test in $\mathbb{Z}_N^*$.

Step 2 The parties perform a Fermat test in the twisted group $\mathbb{T}_N = (\mathbb{Z}_N[x]/(x^2+1))^*/\mathbb{Z}_N^*$. Elements of this group can be viewed as points on the projective line over $\mathbb{Z}_N$. If N is the product of three distinct primes then the order of $\mathbb{T}_N$ is $(p+1)(q+1)(r+1)$. Indeed. x^2+1 is irreducible modulo N since $p = q = r = 3 \bmod 4$. To carry out the Fermat test in $\mathbb{T}_N$ the parties pick a random $g \in \mathbb{T}_N$ and jointly test that $g^{\psi_a+\psi_b+\psi_c} = 1$. If the test fails N is rejected. This step reveals no information other than the outcome of the test.

Step 3 The parties jointly test that N is the product of at most three prime powers. The implementation of this step is explained in the next subsection. If the test fails N is rejected.

Step 4 The parties jointly test that

$$\gcd(N, p+q+r) = 1$$

This step reveals no information other than the outcome of the test. The implementation of this step is explained in the subsection 4.3. If the test fails N is rejected. Otherwise N is accepted as the product of three primes.

The following fact about the twisted group $\mathbb{T}_N = (\mathbb{Z}_N[x]/(x^2+1))^*/\mathbb{Z}_N^*$ is helpful in the proof of the primality test.

Fact 4.1 *Let N be an integer and $k^2|N$ with k prime. Then k divides both $\varphi(N)$ and $|\mathbb{T}_N|$.*

Proof Let $\alpha \geq 2$ be the number of times k divides N. i.e. $N = k^\alpha w$ where $\gcd(k, w) = 1$. Then $\varphi(N) = k^{\alpha-1}(k-1)\varphi(w)$ and hence k divides $\varphi(N)$.

To see that k divides $|\mathbb{T}_N|$ note that $\mathbb{T}_N \cong \mathbb{T}_{k^\alpha} \times \mathbb{T}_w$. When $k = 3 \bmod 4$ we know that x^2+1 is irreducible in $\mathbb{Z}_k$ and hence $|\mathbb{T}_{k^\alpha}| = k^{\alpha-1}(k+1)$. It follows that k divides $|\mathbb{T}_N|$. When $k = 1 \bmod 4$ we have $|\mathbb{T}_{k^\alpha}| = k^{\alpha-1}(k-1)$ and therefore again k divides $|\mathbb{T}_N|$. □

We can now prove that the above four steps are indeed a probabilistic test for proving that N is a product of three primes.

Theorem 4.2 *Let $N = pqr = (p_a+p_b+p_c)(q_a+q_b+q_c)(r_a+r_b+r_c)$ where $p = q = r = 3 \bmod 4$ and* $\gcd(N, p+q+r) = 1$. *If N is a product of three primes it is always accepted. Otherwise, N is rejected with probability at least half. The probability is over the random choices made in steps 1–4 above.*

Proof Suppose p, q and r are distinct primes. Then steps (1),(2) and (3) clearly succeed. Step (4) succeeds by assumption on N. Hence. in this case N always passes the test as required.

Suppose N is not the product of three distinct primes. Assume for a contradiction that N passes all four steps with probability greater than 1/2. Since N passes step (3) with probability greater than 1/2 we know that $N = z_1^{\alpha_1} z_2^{\alpha_2} z_3^{\alpha_3}$ for three primes z_1, z_2, z_3 (not necessarily distinct). Since N passes step (4) we know $\gcd(N, p+q+r) = 1$. Define the following two groups:

$$G = \left\{g \in \mathbb{Z}_N^* \text{ s.t. } g^{\varphi_a+\varphi_b+\varphi_c} = 1\right\}$$
$$H = \left\{g \in \mathbb{T}_N \text{ s.t. } g^{\psi_a+\psi_b+\psi_c} = 1\right\}$$

Clearly G is a subgroup of $\mathbb{Z}_N^*$ and H is a subgroup of the twisted group $\mathbb{T}_N$. We show that at least one of G or H is a proper subgroup which will prove that either steps (1) or (2) fails with probability at least 1/2. There are two cases to consider.

Case 1: p, q, and r are not pairwise relatively prime. By symmetry we may assume, without loss of generality, that $\gcd(p,q) > 1$. Let k be a prime factor of $\gcd(p,q)$. Recall that N is odd so $k > 2$ (since k divides N).

Since $N = pqr$ we know that $k^2|N$. Hence, by Fact 4.1, $k|\varphi(N)$ and $k||\mathbb{T}_N|$. We claim that either k doesn't divide $\hat{\varphi}$ or k doesn't divide $\hat{\psi}$. To see this observe that if $k|\hat{\varphi}$ and $k|\hat{\psi}$, then k divides $\hat{\psi} - \hat{\varphi} = p(2q+2r) + q(2r) + 2$. Since k divides both p and q we conclude that $k|2$, which contradicts $k > 2$.

First we examine when k doesn't divide $\hat{\varphi}$. Since k is a prime factor of $\varphi(N)$ there exists an element $g \in \mathbb{Z}_N^*$ of order k. However, since k does not divide $\hat{\varphi}$ we know that $g^{\hat{\varphi}} \neq 1$. Hence, $g \notin G$ proving that G is a proper subgroup of $\mathbb{Z}_N^*$. If k doesn't divide $\hat{\psi}$ a similar argument proves that H is a proper subgroup of the twisted group $\mathbb{T}_N$.

Case 2: p, q, and r are pairwise relatively prime. We can write $p = z_1^\alpha$, $q = z_2^\beta$ and $r = z_3^\gamma$ with z_1, z_2, z_3 distinct primes. By assumption we know that one of α, β, γ is greater than 1. Without loss of generality we may assume $\alpha > 1$.

We first observe that none of the z_i can divide $\gcd(\hat{\varphi}, \hat{\psi})$. Indeed, if if this were not the case then $z_i | \hat{\varphi} + \hat{\psi} = 2(N + p + q + r)$. But then, since z_i divides N it must also divide $p + q + r$ contradicting the fact that $\gcd(N, p+q+r) = 1$ as tested in step (4).

We now know that z_1 does not divide $\hat{\varphi}$ or it does not divide $\hat{\psi}$. However, since z_1^2 divides N we obtain, by Fact 4.1, that $z_1|\varphi(N)$ and $z_1||\mathbb{T}_N|$. We can now proceed as in case (1) to prove that either G is a proper subgroup of $\mathbb{Z}_N^*$ or H is a proper subgroup of $\mathbb{T}_N$. □

Clearly most integers N that are not a product of three primes will already fail step (1) of the test. Hence, steps (2–4) are most likely executed only once a good candidate N is found.

The condition $\gcd(N, p+q+r) = 1$ is necessary. Without it the theorem is false as can be seen from the following simple example: $p = p_1^3$, $q = ap_1^2 + 1$, $r = bp_1^2 - 1$ where p_1, q, r are three odd primes with $p = q = r = 3 \bmod 4$. In this case $N = pqr$ will always pass steps 1–3 even though it is not a product of three distinct primes.

4.1 Step 3: Testing that $N = p^\alpha q^\beta r^\gamma$

Our protocol for testing that N is a product of three prime powers borrows from a result of van de Graaf and Peralta [12]. Our protocol works as follows:

Step 0 By definition of $\hat{\varphi}$ we know it is divisible by 8. However, the individual shares $\varphi_a, \varphi_b, \varphi_c$ which sum to $\hat{\varphi}$ may not be. To correct this Alice generates two random numbers $a_1, a_2 \in \mathbb{Z}_8$ such that $a_1 + a_2 = \varphi_a \bmod 8$. She sends a_1 to Bob and a_2 to Carol. Alice sets $\varphi_a \leftarrow \varphi_a - a_1 - a_2$, Bob sets $\varphi_b \leftarrow \varphi_b + a_1$ and Carol set $\varphi_c \leftarrow \varphi_c + a_2$. Observe that at this point

$$\frac{\hat{\varphi}}{8} = \frac{\varphi_a}{8} + \left\lfloor \frac{\varphi_b}{8} \right\rfloor + \left\lceil \frac{\varphi_c}{8} \right\rceil$$

Step 1 The parties first agree on eight random numbers $g_1, \dots, g_8$ in $\mathbb{Z}_N^*$, all with Jacobi symbol $+1$.

Step 2 For $i, j = 1, \dots, 8$ we say that i is equivalent to j if

$$\left(\frac{g_i}{g_j}\right)^{\frac{\varphi_a+\varphi_b+\varphi_c}{8}} = 1 \pmod N$$

Since all three parties know g_i and g_j they can test if i is equivalent to j as follows:

1. Alice computes $A = (g_i/g_j)^{\varphi_a/8} \bmod N$.
   Bob computes $B = (g_i/g_j)^{\lfloor \varphi_b/8 \rfloor} \bmod N$ and
   Carol computes $C = (g_i/g_j)^{\lceil \varphi_c/8 \rceil} \bmod N$.

2. Using the comparison protocol of section 3.4 they then test if $ABC = 1 \bmod N$. The comparison protocol reveals no information other than whether $ABC = 1 \bmod N$ or not.

Step 3 If the number of equivalence classes is greater than four N is rejected. Otherwise N is accepted.

Testing that the number of equivalences classes is at most four requires at most 22 invocations of the comparison protocol in the worst case. The reason for restricting attention to elements g_i of Jacobi symbol $+1$ is efficiency. Without this restriction the number of equivalence classes to check for is eight. Thus, many more applications of the comparison protocol are necessary.

The following lemma shows that when N is a product of three distinct primes it is always accepted. When N has more than three prime factors it is rejected with probability at least $1/2$. If N is a product of three prime powers it may always be accepted by this protocol. We use the following notation:

$$\begin{aligned} J &= \{g \in \mathbb{Z}_N^* \text{ s.t. } \left(\frac{g}{N}\right) = +1\} \\ Q &= \{g \in J \text{ s.t. } g \text{ is a quadratic residue in } \mathbb{Z}_N^*\} \end{aligned}$$

The index of Q in J is $2^{d(N)-1}$ or $2^{d(N)}$ where $d(N)$ is the number of distinct prime factors of N.

Lemma 4.3 *Let $N = pqr$ be an integer with $p = q = r = 3 \bmod 4$. If p, q, r are distinct primes then N is always accepted. If the number of distinct prime factors of N is greater than three then N is rejected with probability at least $\frac{1}{2}$.*

Proof If N is the product of three distinct primes then the index of Q in J is four. Two elements $g_1, g_2 \in \mathbb{Z}_N^*$ belong to the same coset of Q in J if and only if g_1/g_2 is a quadratic residue. i.e. if and only if $(g_1/g_2)^{\varphi(N)/8} = 1 \bmod N$. Since in this case $\varphi(N) = \hat{\varphi} = \varphi_a + \varphi_b + \varphi_c$ step (2) tests if g_i and g_j are in the same coset of Q. Since the number of cosets is four there are exactly four equivalence classes and thus N is always accepted.

If N contains at least four distinct prime factors we show that it is rejected with probability at least $1/2$. Define

$$\hat{Q} = \left\{g \in J \text{ s.t. } g^{\hat{\varphi}/8} = 1 \pmod N\right\}$$

Since in this case $\hat{\varphi}$ may not equal $\varphi(N)$ the group $\hat{Q}$ is not the same as the group Q.

We show that the index of $\hat{Q}$ in J is at least eight. Since $p = q = r = 3 \bmod 4$ we know that $\hat{\varphi}/8$ is odd (since $\hat{\varphi} = (p-1)(q-1)(r-1)$). If $g \in J$ satisfies $g^x = 1$ for some odd x then g must be a

quadratic residue (it's root is $g^{(x+1)/2}$). Hence, $\hat{Q} \subseteq Q$ and hence is a subgroup of Q. Since the index of Q in J is at least eight it follows that the index of $\hat{Q}$ in J is at least eight.

It remains to show that when the index of $\hat{Q}$ in J is at least eight then N is rejected with probability at least 1/2. In step (2) two elements $g_1, g_2 \in J$ are equivalent if they belong to the same coset of $\hat{Q}$ in J. Let R be the event that all 8 elements $g_i \in J$ chosen randomly in step (1) fall into only four of the eight cosets. Then

$$\Pr[R] \leq \binom{8}{4} \cdot \left(\frac{1}{2}^8\right) = 0.27 < \frac{1}{2}$$

N is accepted only when the event R occurs. Since it occurs with probability less than 1/2 the number N is rejected with probability at least 1/2 as required. □

Next we prove that the protocol leaks no information when N is indeed the product of three distinct primes. In case N is not of this form the protocol may leak some information; however in this case N is discarded and is of no interest. To prove that the protocol leaks no information we rely on a classic cryptographic assumption [4] called *Quadratic Residue Indistinguishability* or QRI for short. This cryptographic assumption states that when $N = pq$ with $p = q = 3 \bmod 4$ no polynomial time algorithm can distinguish between the groups J and Q defined above. In other words, for any polynomial time algorithm $\mathcal{A}$ and any constant $c > 0$

$$\left| \Pr_{g \in J}[\mathcal{A}(g) = \text{"yes"}] - \Pr_{g \in Q}[\mathcal{A}(g) = \text{"yes"}] \right| < \frac{1}{(\log N)^c}$$

The following lemma relies on QRI when N is the product of *three* primes.

Lemma 4.4 *If N is a product of three distinct primes then the protocol is 1-private assuming QRI.*

Proof Sketch To prove that each party learns no information other than the fact that N is a product of three prime powers we provide a simulation argument. We show that each party can simulate its view of the protocol. Hence, whatever values it receives from its peers, it could have generated itself. By symmetry we may only consider Alice. Alice's view of the protocol consists of the elements $g_1, \ldots, g_8$ and bit values $b_{i,j}$ indicating whether $(g_i/g_j)^{\varphi} = 1$. (we already gave a simulation algorithm for the comparison protocol in Section 3.4). Thus, Alice learns whether g_i/g_j is a quadratic residue or not. We argue that under QRI this provides no computational information since it can be simulated. To simulate Alice's view the simulation algorithm works as follows: it picks eight random elements $g_1, \ldots, g_8 \in J$. It then randomly associates with each g_i a value in the set $\{0, 1, 2, 3\}$. This value represents the coset of Q that g_i is in. The simulator then says that g_i/g_j is a quadratic residue if and only if the value associates with g_i is equal to that associated with g_j. Under QRI the resulting distribution on $g_1, \ldots, g_8, b_{1,1}, \ldots, b_{8,8}$ is computationally indistinguishable from Alice's true view of the protocol. We note that the value $a_1 \in [0, 8]$ Alice sends Bob in Step (0) is a uniform random element of $\mathbb{Z}_8$. Hence, it is trivially simulatable by Bob. Similarly $a_2 \in [0, 8]$ is simulatable by Carol. □

4.2 Implementing a Fermat test with no information leakage

We briefly show how to implement a Fermat test in $\mathbb{Z}_N^*$ without leaking any extra information about the private shares. The exact same method works in the twisted group $\mathbb{T}_N$ as well.

To check that $g \in \mathbb{Z}_N^*$ satisfies $g^{\varphi_a+\varphi_b+\varphi_c} = 1 \bmod N$ we perform the following steps:

Step 1 Each party computes $R_i = g^{\varphi_i} \bmod N$ for $i = a, b, c$.

Step 2 They test that $R_a R_b R_c = 1 \bmod N$ be revealing the values R_1, R_2, R_3. Accept N if the test succeeds. Otherwise reject.

Clearly the protocol succeeds if and only if $g^{\varphi} = 1 \bmod N$. We show that it leaks no other information.

Lemma 4.5 *If $N = pqr$ is the product of three distinct primes then the protocol is 2−private.*

Proof We show that any two parties learn no information about the private share of the third other than $g^{\varphi} = 1 \bmod N$. By symmetry we restrict attention to Alice and Bob. Since by assumption N is the product of three primes we know that $g^{\varphi} = 1 \bmod N$. Hence, $g^{\varphi_a + \varphi_b} = g^{-\varphi_c}$. To simulate the value received from Carol the simulation algorithm simply computes $g^{-\varphi_c}$. Indeed, this is a perfect simulation of Alice and Bob's view. Thus, they learn nothing from Carol's message since they could have generated it themselves. □

4.3 Step 4: Testing that $\gcd(N, p+q+r) = 1$ in zero knowledge

Our protocol for this step is based on a protocol similar to the one used in the computation of N. We proceed as follows:

Step 1 Alice picks a random $y_a \in Z_N$. Bob picks a random $y_b \in \mathbb{Z}_N$. Carol picks a random $y_c \in \mathbb{Z}_N$.

Step 2 Using the BGW protocol as in Section 3.2 they compute

$$R = (p_a + q_a + p_b + q_b + p_c + q_c)(y_a + y_b + y_c) \bmod N$$

At the end of the protocol R is publicly known. however no other information about the private shares is revealed.

Step 3 Now that R is public the parties test that $\gcd(R, N) = 1$. If not, N is rejected. Otherwise N is accepted.

Lemma 4.6 *If $N = pqr$ is the product of three distinct n-bit primes with $\gcd(N, p+q+r) = 1$ then N is accepted with probability $1 - \epsilon$ for $\epsilon < 1/2^n$. Otherwise, N is always rejected.*

Proof Clearly if $\gcd(N, p+q+r) > 1$ then $\gcd(R, N) > 1$ and therefore N is always rejected. If $\gcd(N, p+q+r) = 1$ then N is rejected only if $\gcd(N, y_a + y_b + y_c) > 1$. Since $y_a + y_b + y_c$ is a random element of $\mathbb{Z}_N$ this happens with probability less than $(1/2)^n$. □

Lemma 4.7 *If $N = pqr$ is the product of three distinct n-bit primes with $\gcd(N, p+q+r) = 1$ then the protocol is 1−private.*

Proof Since the BGW protocol is 1−private the above protocol can be at most 1−private. We show how to simulate Alice's view. Alice's view consists of her private shares p_a, q_a, y_a and the number R. Since R is independent of her private shares the simulator can simulate Alice's view by simply picking R in $\mathbb{Z}_N$ at random. This is a perfect simulation. □

5 Extensions

One can naturally extend our protocols in two ways. First, one may allow more than three parties to generate a product of three primes with an unknown factorization. Second, one may wish to design primality tests for testing that N is a product of k primes for some small k. We briefly discuss both extensions below.

Our protocols easily generalize to allow any number of parties. When k parties are involved the protocols can be made $\lfloor\frac{k-1}{2}\rfloor$ private. This is optimal in the information theoretic sense and follows from the privacy properties of the BGW protocol. The only complexities in this extension are the comparison protocol of Section 3.4 and Step (0) of Section 4.1. Both protocols generalize to k parties however they require a linear (in k) number of rounds of communication.

Securely testing that N is a product of k primes for some fixed $k > 3$ seems to be harder. Our results apply when $k = 4$ (indeed Theorem 4.2 remains true in this case). For $k > 4$ more complex algorithms are necessary. This extension may not be of significant interest since it is not well motivated and requires complex protocols.

Another natural question is whether only two parties can generate a product of three primes with an unknown factorization. The answer appears to be yes although the protocols cannot be information theoretically secure. Essentially one needs to replace the BGW protocol for computing N with a two-party private multiplication protocol. This appears to be possible using results of [6, 3].

6 Conclusions and open problems

Our main contribution is the design of a probabilistic primality test that enables three (or more) parties to generate a number N with an unknown factorization and test that N is the product of three distinct primes. The correctness of our primality test relies on the fact that we simultaneously work in two different subgroups of $\mathbb{Z}_N[x]/(x^2+1)^*$, namely $\mathbb{Z}_N^*$ and the projective line over $\mathbb{Z}_N$. Our protocol generalizes to an arbitrary number of parties k and achieves $\lfloor\frac{k-1}{2}\rfloor$ privacy – the best possible in an information theoretic setting.

Recall that our primality test can be applied to $N = pqr$ whenever $p = q = r = 3 \bmod 4$. We note that simple modifications enable one to apply the test when $p = q = r = 1 \bmod 4$ (essentially this is done by reversing the roles of $\mathbb{Z}_N^*$ and the twisted group). However, it seems that one of these restrictions is necessary. We do not know how to carry out the test without the assumption that $p = q = r \bmod 4$. The assumption plays a crucial role in the proof of Lemma 4.3.

A natural question is whether more advanced primality testing techniques can be used to improve the efficiency of our test. For instance, recent elegant techniques due to Grantham [11] may be applicable in our scenario as well.

References

[1] M. Ben-Or, S. Goldwasser, A. Wigderson, "Completeness theorems for non-cryptographic fault tolerant distributed computation", STOC 1988, pp. 1–10.

[2] D. Boneh, M. Franklin, "Efficient generation of shared RSA keys", in Proceedings Crypto' 97, pp. 425–439.

[3] D. Boneh. M. Franklin, "A heuristic for two party generation of shared RSA keys", unpublished manuscript.

[4] M. Blum. S. Goldwasser. "An efficient probabilistic public key encryption scheme that hides all partial information", in Proceedings Crypto 84. pp. 289–302.

[5] D. Chaum. C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols," ACM STOC 1988, 11-19.

[6] C. Cocks. "Split knowledge generation of RSA parameters". Available from the author cliff_cocks@cesg.gov.uk.

[7] R. Fagin. M. Naor, P. Winkler, "Comparing information without leaking it". CACM, Vol 39, No. 5. May 1996. pp. 77–85.

[8] Y. Frankel, "A practical protocol for large group oriented networks", Eurocrypt 89, pp. 56–61.

[9] Y. Frankel, P. MacKenzie, M. Yung, "Robust efficient distributed RSA key generation", Preprint.

[10] P. Gemmel. "An introduction to threshold cryptography", in CryptoBytes, a technical newsletter of RSA Laboratories. Vol. 2. No. 7. 1997.

[11] J. Grantham, "A probable prime test with high confidence", http://www.clark.net/pub/grantham/pseudo/

[12] J. van de Graaf. Rene Peralta, "A simple and secure way to show the validity of your public key". in Proc. Crypto 87, pp. 128–134.

[13] A. Lenstra. H.W. Lenstra ed.. "The development of the number field sieve". Springer-Verlag, LNCS 1554. 1994.

[14] H. W. Lenstra. "Factoring integers with elliptic curves". Annals of Math.. Vol. 126. 1987. pp. 649–673.

[15] A Shamir. "How to share a secret". Comm. of the ACM, Vol. 22. No. 11. Nov. 1979, pp. 612–613.

[16] M. Wiener. "Cryptanalysis of short RSA secret exponents", IEEE Transactions on Info. Th., Vol. 36. No. 3, 1990, pp. 553–558.

[17] A. Yao. "How to generate and exchange secrets". FOCS 1986. pp. 162-167.

1982 INTERNATIONAL SYMPOSIUM ON
INFORMATION THEORY

LES ARCS, FRANCE
JUNE 21-25, 1982

Sponsored by:
The Institute of Electrical and Electronics Engineers,
Information Theory Group

Co-sponsored by:
Union Radio Scientifique Internationale
Société Française de Théorie de l'Information

Co-Chairmen:

C.W. Helstrom — B. Picinbono

Vice-Chairmen:

N. Cot — R. Gray

International Advisory Committee:

A. Viterbi (Chairman) (U.S.A.)
G. Battail (France)
J. Bezerra (Brazil)
I. Boxma (The Netherlands)
B. Dorsch (F.R. Germany)
I. Ingemarsson (Sweden)
J. Hayes (Canada)
K. Kobayashi (Japan)
K. Kozlowski (Poland)
A. Lempel (Israel)
K. Marton (Hungary)
V. Milutinovic (Yugoslavia)
E. Oja (Finland)
B. Patek (Czechoslovakia)
G. Tartara (Italy)
N. Tepedelenlioglu (Turkey)
G. Ungerboeck (Switzerland)
E. Van der Meulen (Belgium)
M. Wu (China)
K. Zigangirov (USSR)

Program Committee:

J.M. Goethals (Chairman)
R. Ahlswede
P. Brémaud
P. Camion
P. Devijver
P. Flajolet
G. Longo
O. Macchi
J. Massey
M. Métivier
E. Van der Meulen
J. Schalkwijk

Finance: L. Milstein (Treasurer), S. Rimoldi
Registration: S. Rimoldi
Publicity: J. Dunham, D. Neuhoff
Publications: S. Harari
Local arrangements: N. Cot (Chairman), G. Cohen, B. Bouchon, S. Harari, P. Brémaud, O. Macchi, P. Camion

IEEE Catalog Number 82CH1767-3 IT

Library of Congress Catalog Card Number 72-179437

SESSION E4
Cryptography (1)

SESSION E5
Complexity

* Denotes long papers.

Here $\bar{y}$ is an unknown integer vector. The matrices K and L and the vector $\bar{S}$ are known. The inequality $0 \leq x_i \leq 1$ is used to find integer solutions for $\bar{y}$, which are inserted in the last equation, yielding $\bar{x}$ which are tested as hypothetical solutions to the knapsack problem.

A FAST GENERATOR OF LARGE PRIME NUMBERS, J.J. Quisquater and C. Couvreur (Philips Research Laboratory, Av. van Beceleare 2, Box 8, B-1170 Brussels, Belgium). An analysis of the Rivest-Shamir-Adleman public-key cryptosystem has shown that its security is based on the difficulty of factoring numbers r which are the product of two large primes p and q. Some constraints must be put on p and q to create secure keys. The process of devising suitable values for p and q requires first a method for finding large random primes (each about 10^{50} if p and q must be about 10^{100}).

A fast algorithm for generating such primes is presented. This algorithm is distinguishable from the probabilistic algorithms in that the so generated numbers are certified to be prime for a lower time complexity. Recently, Adleman, Rumely, Pomerance and Lenstra described an algorithm for distinguishing prime numbers from composite numbers. On the other hand, Williams and Schmid, Crandall and Penk, and chiefly Plaisted proposed a method for generating prime numbers. These have been adapted for our purposes.

An experimental version of our generator has been implemented on a VAX 11-780. The following table gives some results on the generation of random prime numbers of given length. The *likely primes* have been found using Rabin's algorithm, each prime being tested with 20 random numbers. The *certified primes* are generated by our algorithm. Let us remark that the comparisons are made with the same sets numbers to be tested and the same partial sieving.

length number (decimal representation)	certified prime	likely prime
50	5 sec.	20 sec.
67	11 sec.	50 sec.
115	60 sec.	250 sec.
200	400 sec.	1800 sec.

ON THE DESIGN AND ANALYSIS OF NEW CIPHER SYSTEMS RELATED TO THE DES, I. Schaumüller-Bichl (Institute of Systems Science, Johannes Kepler Universität Linz, A-4040 Linz, Austria). One essential, but up to now almost neglected item of the DES is its basic encryption scheme. It warrants - independently of the heavily discussed specific choices of the S-boxes, permutations,... - that for every encipher function there exists an easy to find decipher function and furthermore, that the algorithm is secure against "backward computation".

Following this idea in the first part of the paper a new cipher system, called C80, is presented which is based on the DES encryption scheme. The components of the key are selected from the set of the residue classes modulo m, whereas m depends on the variable block length. The cipher function f is based on computing the scalar product of the key and parts of the plaintext. C80 provides any required level of security against brute force attacks and also promises to resist a short cut attack even better than the DES does.

The "Generalized DES scheme" (GDES scheme) presented in the second chapter is an attempt to generalize the DES encryption scheme in a way that permits multiple encryption speed without risking security.

CRITICAL ANALYSIS OF THE SECURITY OF KNAPSACK PUBLIC KEY ALGORITHMS, Y. Desmedt, J. Vandewalle and R. Govaerts (Katholieke Universiteit Leuven, Department Electrotechniek, Afdeling E.S.T.A., Kardinaal Mercierlaan 94, B-3030 Heverlei, Belgium). The authors claim that the security of the Merkle-Hellman algorithm is greatly exaggerated. They show that for their enciphering keys there exist infinitely many superincreasing keys which can decipher all messages. For example, applying the transformation x 46 mod 77 to the enciphering key (5457, 1663, 216, 6013, 7439) of Merkle and Hellman, one obtains the sequence (2, 37, 3, 14, 6), which is superincreasing after reordering.

Moreover, an iterative transformation x w mod m in the construction of the enciphering key may not increase the security. For example (25, 87, 33) is an enciphering key which is obtained after 2 transformations from (5, 10, 20) and hence considered to be safer by Merkle and Hellman. This enciphering key is however totally insecure because it is already superincreasing after reordering.

The effect of such transformations can be reformulated as a x w-s x m for some s. This shows that several intervals of w/m lead to useful transformations. Moreover one key can decipher all messages if m is larger than the sum of all numbers in the deciphering keys, allowing us to generalize and improve the cracking idea and easy deciphering key of Herlestam and Shamir

HANDBOOK of APPLIED CRYPTOGRAPHY

Alfred J. Menezes
Paul C. van Oorschot
Scott A. Vanstone

CRC Press
Boca Raton New York London Tokyo

Library of Congress Cataloging-in-Publication Data

Menezes, A. J. (Alfred J.), 1965–
Handbook of applied cryptography / Alfred Menezes, Paul van Oorschot, Scott Vanstone.
p. cm. -- (CRC Press series on discrete mathematics and its applications)
Includes bibliographical references and index.
ISBN 0-8493-8523-7 (alk. paper)
1. Computers--Access control--Handbooks, manuals, etc.
2. Cryptography--Handbooks, manuals, etc. I. Van Oorschot, Paul C.
II. Vanstone, Scott A. III. Title. IV. Series: Discrete mathematics and its applications.
QA76.9.A25M463 1996
005.8¢2--dc20 96-27609
CIP

International Standard Book Number 0-8493-8523-7
Library of Congress Card Number 96-27609
Printed in the United States of America 3 4 5 6 7 8 9 0
Printed on acid-free paper

Informally speaking, if $A \equiv_P B$ then A and B are either both tractable or both intractable, as the case may be.

Chapter outline

The remainder of the chapter is organized as follows. Algorithms for the integer factorization problem are studied in §3.2. Two problems related to factoring, the RSA problem and the quadratic residuosity problem, are briefly considered in §3.3 and §3.4. Efficient algorithms for computing square roots in $\mathbb{Z}_p$, p a prime, are presented in §3.5, and the equivalence of the problems of finding square roots modulo a composite integer n and factoring n is established. Algorithms for the discrete logarithm problem are studied in §3.6, and the related Diffie-Hellman problem is briefly considered in §3.7. The relation between the problems of factoring a composite integer n and computing discrete logarithms in (cyclic subgroups of) the group $\mathbb{Z}_n^*$ is investigated in §3.8. The tasks of finding partial solutions to the discrete logarithm problem, the RSA problem, and the problem of computing square roots modulo a composite integer n are the topics of §3.9. The L^3-lattice basis reduction algorithm is presented in §3.10, along with algorithms for the subset sum problem and for simultaneous diophantine approximation. Berlekamp's Q-matrix algorithm for factoring polynomials is presented in §3.11. Finally, §3.12 provides references and further chapter notes.

3.2 The integer factorization problem

The security of many cryptographic techniques depends upon the intractability of the integer factorization problem. A partial list of such protocols includes the RSA public-key encryption scheme (§8.2), the RSA signature scheme (§11.3.1), and the Rabin public-key encryption scheme (§8.3). This section summarizes the current knowledge on algorithms for the integer factorization problem.

3.3 Definition The *integer factorization problem* (FACTORING) is the following: given a positive integer n, find its prime factorization; that is, write $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ where the p_i are pairwise distinct primes and each $e_i \geq 1$.

3.4 Remark (*primality testing vs. factoring*) The problem of *deciding* whether an integer is composite or prime seems to be, in general, much easier than the factoring problem. Hence, before attempting to factor an integer, the integer should be tested to make sure that it is indeed composite. Primality tests are a main topic of Chapter 4.

3.5 Remark (*splitting vs. factoring*) A *non-trivial factorization* of n is a factorization of the form $n = ab$ where $1 < a < n$ and $1 < b < n$; a and b are said to be *non-trivial factors* of n. Here a and b are not necessarily prime. To solve the integer factorization problem, it suffices to study algorithms that *split* n, that is, find a non-trivial factorization $n = ab$. Once found, the factors a and b can be tested for primality. The algorithm for splitting integers can then be recursively applied to a and/or b, if either is found to be composite. In this manner, the prime factorization of n can be obtained.

3.6 Note (*testing for perfect powers*) If $n \geq 2$, it can be efficiently checked as follows whether or not n is a *perfect power*, i.e., $n = x^k$ for some integers $x \geq 2$, $k \geq 2$. For each prime

x	$v(x)$	x	$v(x)$	x	$v(x)$	x	$v(x)$	x	$v(x)$
0	(000)	6	(001)	12	(002)	18	(003)	24	(004)
1	(111)	7	(112)	13	(113)	19	(114)	25	(110)
2	(022)	8	(023)	14	(024)	20	(020)	26	(021)
3	(103)	9	(104)	15	(100)	21	(101)	27	(102)
4	(014)	10	(010)	16	(011)	22	(012)	28	(013)
5	(120)	11	(121)	17	(122)	23	(123)	29	(124)

Table 14.14: *Modular representations (see Example 14.69).*

$v_1^d \bmod p$ and $v_2^d \bmod q$ is faster than computing $x^d \bmod n$. For RSA, if p and q are part of the private key, modular representation can be used to improve the performance of both decryption and signature generation (see Note 14.75).

Converting an integer x from a base b representation to a modular representation is easily done by applying a modular reduction algorithm to compute $v_i = x \bmod m_i$, $1 \le i \le t$. Modular representations of integers in $\mathbb{Z}_M$ may facilitate some computational efficiencies, provided conversion from a standard radix to modular representation and back are relatively efficient operations. Algorithm 14.71 describes one way of converting from modular representation back to a standard radix representation.

14.5.2 Garner's algorithm

Garner's algorithm is an efficient method for determining x, $0 \le x < M$, given $v(x) = (v_1, v_2, \dots, v_t)$, the residues of x modulo the pairwise co-prime moduli $m_1, m_2, \dots, m_t$.

14.71 Algorithm Garner's algorithm for CRT

INPUT: a positive integer $M = \prod_{i=1}^{t} m_i > 1$, with $\gcd(m_i, m_j) = 1$ for all $i \neq j$, and a modular representation $v(x) = (v_1, v_2, \dots, v_t)$ of x for the m_i.
OUTPUT: the integer x in radix b representation.

1. For i from 2 to t do the following:
   - 1.1 $C_i \leftarrow 1$.
   - 1.2 For j from 1 to $(i-1)$ do the following:
     - $u \leftarrow m_j^{-1} \bmod m_i$ (use Algorithm 14.61).
     - $C_i \leftarrow u \cdot C_i \bmod m_i$.
2. $u \leftarrow v_1$, $x \leftarrow u$.
3. For i from 2 to t do the following: $u \leftarrow (v_i - x)C_i \bmod m_i$, $x \leftarrow x + u \cdot \prod_{j=1}^{i-1} m_j$.
4. Return(x).

14.72 Fact x returned by Algorithm 14.71 satisfies $0 \le x < M$, $x \equiv v_i \pmod{m_i}$, $1 \le i \le t$.

14.73 Example (*Garner's algorithm*) Let $m_1 = 5$, $m_2 = 7$, $m_3 = 11$, $m_4 = 13$, $M = \prod_{i=1}^{4} m_i = 5005$, and $v(x) = (2,1,3,8)$. The constants C_i computed are $C_2 = 3$, $C_3 = 6$, and $C_4 = 5$. The values of (i, u, x) computed in step 3 of Algorithm 14.71 are $(1,2,2)$, $(2,4,22)$, $(3,7,267)$, and $(4,5,2192)$. Hence, the modular representation $v(x) = (2,1,3,8)$ corresponds to the integer $x = 2192$. □

14.74 Note (*computational efficiency of Algorithm 14.71*)

(i) If Garner's algorithm is used repeatedly with the same modulus M and the same factors of M, then step 1 can be considered as a precomputation, requiring the storage of $t-1$ numbers.

(ii) The classical algorithm for the CRT (Algorithm 2.121) typically requires a modular reduction with modulus M, whereas Algorithm 14.71 does not. Suppose M is a kt-bit integer and each m_i is a k-bit integer. A modular reduction by M takes $O((kt)^2)$ bit operations, whereas a modular reduction by m_i takes $O(k^2)$ bit operations. Since Algorithm 14.71 only does modular reduction with m_i, $2 \le i \le t$, it takes $O(tk^2)$ bit operations in total for the reduction phase, and is thus more efficient.

14.75 Note (*RSA decryption and signature generation*)

(i) (*special case of two moduli*) Algorithm 14.71 is particularly efficient for RSA moduli $n = pq$, where $m_1 = p$ and $m_2 = q$ are distinct primes. Step 1 computes a single value $C_2 = p^{-1} \bmod q$. Step 3 is executed once: $u = (v_2 - v_1)C_2 \bmod q$ and $x = v_1 + up$.

(ii) (*RSA exponentiation*) Suppose p and q are t-bit primes, and let $n = pq$. Let d be a $2t$-bit RSA private key. RSA decryption and signature generation compute $x^d \bmod n$ for some $x \in \mathbb{Z}_n$. Suppose that modular multiplication and squaring require k^2 bit operations for k-bit inputs, and that exponentiation with a k-bit exponent requires about $\frac{3}{2}k$ multiplications and squarings (see Note 14.78). Then computing $x^d \bmod n$ requires about $\frac{3}{2}(2t)^3 = 12t^3$ bit operations. A more efficient approach is to compute $x^{d_p} \bmod p$ and $x^{d_q} \bmod q$ (where $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$), and then use Garner's algorithm to construct $x^d \bmod pq$. Although this procedure takes two exponentiations, each is considerably more efficient because the moduli are smaller. Assuming that the cost of Algorithm 14.71 is negligible with respect to the exponentiations, computing $x^d \bmod n$ is about $\frac{3}{2}(2t)^3/2(\frac{3}{2}t^3) = 4$ times faster.

14.6 Exponentiation

One of the most important arithmetic operations for public-key cryptography is exponentiation. The RSA scheme (§8.2) requires exponentiation in $\mathbb{Z}_m$ for some positive integer m, whereas Diffie-Hellman key agreement (§12.6.1) and the ElGamal encryption scheme (§8.4) use exponentiation in $\mathbb{Z}_p$ for some large prime p. As pointed out in §8.4.2, ElGamal encryption can be generalized to any finite cyclic group. This section discusses methods for computing the *exponential* g^e, where the *base* g is an element of a finite group G (§2.5.1) and the *exponent* e is a non-negative integer. A reader uncomfortable with the setting of a general group may consider G to be $\mathbb{Z}_m^*$; that is, read g^e as $g^e \bmod m$.

An efficient method for multiplying two elements in the group G is essential to performing efficient exponentiation. The most naive way to compute g^e is to do $e-1$ multiplications in the group G. For cryptographic applications, the order of the group G typically exceeds 2^{160} elements, and may exceed 2^{1024}. Most choices of e are large enough that it would be infeasible to compute g^e using $e-1$ successive multiplications by g.

There are two ways to reduce the time required to do exponentiation. One way is to decrease the time to multiply two elements in the group; the other is to reduce the number of multiplications used to compute g^e. Ideally, one would do both.

This section considers three types of exponentiation algorithms.

UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): COLLINS et al. Patent No. 5,848,159

Docket No. 20206-014(PT-TA-410) By: LS/jmp

Issued: December 8, 1998

For: **PUBLIC KEY CRYPTOGRAPHIC APPARATUS AND METHOD**

The stamp of the U.S. Patent and Trademark Office hereon acknowledges receipt of the following:

1. Assignment Recordation Sheet;
2. Articles of Merger; and
3. Check No. 124498 for $40.00.

JC914 U.S. PTO 09/694416 10/20/00

OPPENHEIMER

OPPENHEIMER WOLFF & DONNELLY LLP
Palo Alto, CA 94304

17-2 / 910

124498

Date 10/12/00

Amount

*** *** *40.00

Exactly FORTY AND 00/100 DOLLARS

The Order Of ASSISTANT COMMISSIONER OF PATENTS AND TRADEMARKS

USbank.

⑈124498⑈ ⑆091000022⑆ 173102134785⑈

Form PTO-1595 | U.S. Department of Commerce

RECORDATION FORM COVER SHEET
PATENTS ONLY

Docket No. 20206-014(PT-TA-410)

To the Hon. Commissioner of Patents & Trademarks: Please record the attached original documents or copy thereof.

1. Name of conveying party(ies):

Tandem Computers Incorporated

Additional name of conveying party attached? ☐ Yes ☑ No

2. Name and address of receiving party(ies):

Name: **Compaq Computer Corporation**

Address: **P.O. Box 692000**
20555 SH 249
City: **Houston** State: **TX** ZIP: **77070-2698**

Additional name(s) and address(es) attached? ☐ Yes ☑ No

3. Nature of conveyance:

☑ Articles of Merger of Parent and Subsidiary Corporations

Execution Date: December 31, 1998

4. Application number(s) or patent numbers:

If this document is being filed together with a new application, the execution date of the application is:

A. Patent Application No.(s)

B. Patent No.(s)

5,848,159

Additional numbers attached? ☐ Yes ☑ No

5. Name and address party to whom correspondence concerning document should be mailed:

Name: Leah Sherry
Firm: Oppenheimer Wolff & Donnelly LLP

Street Address: 1400 Page Mill Rd.
City: Palo Alto State: California ZIP: 94304

6. Total number of applications and patents involved: 1

7. Total fee (37 CFR 3.41) $40.00

☑ Enclosed

☑ Any discrepancy or overpayment is authorized to be charged to deposit account

8. Deposit Account number: 02-3964

(Attach duplicate copy of this page if paying by deposit account)

DO NOT USE THIS SPACE

9. Statement and signature.

To the best of my knowledge and belief, the foregoing information is true and correct and any attached copy is a true copy of the original document.

Leah Sherry		October 12, 2000
Name of Person Signing	Signature	Date

Total number of pages including cover sheet, attachments, and document: 4

SV: 108748 v01 10/12/2000

State of Delaware

Office of the Secretary of State

I, EDWARD J. FREEL, SECRETARY OF STATE OF THE STATE OF DELAWARE, DO HEREBY CERTIFY THE ATTACHED IS A TRUE AND CORRECT COPY OF THE CERTIFICATE OF OWNERSHIP, WHICH MERGES:

"TANDEM COMPUTERS, INCORPORATED", A DELAWARE CORPORATION,

WITH AND INTO "COMPAQ COMPUTER CORPORATION" UNDER THE NAME OF "COMPAQ COMPUTER CORPORATION", A CORPORATION ORGANIZED AND EXISTING UNDER THE LAWS OF THE STATE OF DELAWARE, AS RECEIVED AND FILED IN THIS OFFICE THE TWENTY-SECOND DAY OF DECEMBER, A.D. 1998, AT 4:30 O'CLOCK P.M.

AND I DO HEREBY FURTHER CERTIFY THAT THE EFFECTIVE DATE OF THE AFORESAID CERTIFICATE OF OWNERSHIP IS THE THIRTY-FIRST DAY OF DECEMBER, A.D. 1998.

A FILED COPY OF THIS CERTIFICATE HAS BEEN FORWARDED TO THE NEW CASTLE COUNTY RECORDER OF DEEDS.

Edward J. Freel

Edward J. Freel, Secretary of State

0932025 8100M

981497477

AUTHENTICATION: 9482768

DATE: 12-23-98

CERTIFICATE OF OWNERSHIP AND MERGER

MERGING

TANDEM COMPUTERS INCORPORATED

INTO

COMPAQ COMPUTER CORPORATION

* * * *

Compaq Computer Corporation, a corporation organized and existing under the laws of Delaware,

DOES HEREBY CERTIFY:

FIRST: That this corporation was incorporated on the 16th day of February, 1982, pursuant to the General Corporation Laws of the State of Delaware.

SECOND: That this corporation owns all of the outstanding shares of each class of the stock of Tandem Computers Incorporated, a corporation incorporated on the 7th day of January, 1980, pursuant to the General Corporation Laws of the State of Delaware.

THIRD: That this corporation, by the following resolutions of its Board of Directors, duly adopted at a meeting held on the 10th day of December, 1998, determined to and did merge into itself said Tandem Computers Incorporated:

> RESOLVED, that the merger of Tandem Computers Incorporated into the Company be and it hereby is approved, and Compaq Computer Corporation does hereby assume all of the liabilities and obligations of and merge into itself Tandem Computers Incorporated;
>
> FURTHER RESOLVED, that the merger shall become effective on midnight December 31, 1998; and

FURTHER RESOLVED, that any Vice President of the Company be and hereby is authorized and directed to execute a Certificate of Ownership and Merger setting forth a copy of the foregoing resolutions and to cause same to be filed with the Secretary of State, and to take such further actions and to execute such documents as may be necessary to implement the merger.

IN WITNESS WHEREOF, said Compaq Computer Corporation has caused this Certificate to be signed by Linda S. Auwers, its Vice President, Associate General Counsel and Secretary, this 22nd day of December, 1998.

By Linda S. Auwers

Linda S. Auwers
Vice President, Associate General
Counsel and Secretary

( )5 77-04040045

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
ASSISTANT SECRETARY AND COMMISSIONER
OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

JULY 15, 1997 97 JUL 22 AM 9:59

PTAS

TOWNSEND AND TOWNSEND AND CREW LLP
ROBERT J. BENNETT
TWO EMBARCADERO CENTER, 8TH FLOOR
SAN FRANCISCO, CA 94111-3834

100436861A

JC914 U.S. PTO
09/694416
10/20/00

UNITED STATES PATENT AND TRADEMARK OFFICE
NOTICE OF RECORDATION OF ASSIGNMENT DOCUMENT

THE ENCLOSED DOCUMENT HAS BEEN RECORDED BY THE ASSIGNMENT DIVISION OF THE U.S. PATENT AND TRADEMARK OFFICE. A COMPLETE MICROFILM COPY IS AVAILABLE AT THE ASSIGNMENT SEARCH ROOM ON THE REEL AND FRAME NUMBER REFERENCED BELOW.

PLEASE REVIEW ALL INFORMATION CONTAINED ON THIS NOTICE. THE INFORMATION CONTAINED ON THIS RECORDATION NOTICE REFLECTS THE DATA PRESENT IN THE PATENT AND TRADEMARK ASSIGNMENT SYSTEM. IF YOU SHOULD FIND ANY ERRORS OR HAVE QUESTIONS CONCERNING THIS NOTICE, YOU MAY CONTACT THE EMPLOYEE WHOSE NAME APPEARS ON THIS NOTICE AT 703-308-9723. PLEASE SEND REQUEST FOR CORRECTION TO: U.S. PATENT AND TRADEMARK OFFICE, ASSIGNMENT DIVISION, BOX ASSIGNMENTS, NORTH TOWER BUILDING, SUITE 10C35, WASHINGTON, D.C. 20231.

RECORDATION DATE: 05/07/1997 REEL/FRAME: 8542/0875
NUMBER OF PAGES: 4

BRIEF: ASSIGNMENT OF ASSIGNOR'S INTEREST (SEE DOCUMENT FOR DETAILS).

ASSIGNOR:
COLLINS, THOMAS DOC DATE: 04/29/1997

ASSIGNOR:
HOPKINS, DALE DOC DATE: 04/29/1997

ASSIGNOR:
LANGFORD, SUSAN DOC DATE: 04/30/1997

ASSIGNOR:
SABIN, MICHAEL DOC DATE: 04/30/1997

ASSIGNEE:
TANDEM COMPUTERS INCORPORATED
10435 NORTH TANTAU AVENUE
CUPERTINO, CALIFORNIA 95014

SERIAL NUMBER: 08784453 FILING DATE: 01/16/1997
PATENT NUMBER: ISSUE DATE:

8542/0875 PAGE 2

SEDLEY PYNE, EXAMINER
ASSIGNMENT DIVISION
OFFICE OF PUBLIC RECORDS

09694410.102000

06-16-1997

Attorney Docket No. 10577-404

FORM PTO-1595
(Rev. 6-93)

U.S. DEPARTMENT OF COMMERCE
Patent and Trademark Office

100436861

RECEIVED
MAY 07 1997
RECEIPT ACCTING. DIV.

To the Honorable Asst. Commissioner for ... al documents or copy thereof.

1. Name of conveying party(ies): MED 5-7-97

Thomas Collins
Dale Hopkins
Susan Langford
Michael Sabin

Additional name(s) of conveying party(ies) attached? ☐ Yes ☒ No

2. Name and address of receiving party(ies):

Name: Tandem Computers Incorporated

Internal Address:

Street Address: 10435 North Tantau Avenue

City: Cupertino State: California ZIP: 95014

Additional name(s) & address(es) attached? ☐ Yes ☒ No

3. Nature of conveyance:

☒ Assignment ☐ Merger
☐ Security Agreement ☐ Change of Name
☐ Other:

Execution Date: 4/29/97 and 4/30/97

4. Application number(s) or patent number(s).

If this document is being filed together with a new application, the execution date of the application is:

A. Patent Application No.(s) 08/784,453

B. Patent No.(s)

Additional numbers attached? ☐ Yes ☒ No

5. Name and address of party to whom correspondence concerning document should be mailed:

Name: Robert J. Bennett
TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, 8th Floor
San Francisco, California 94111-3834
(415) 576-0200

6. Total number of applications and patents involved: 1

7. Total fee (37 CFR 3.41):................ $ 40.00

☐ Enclosed ☒ Charge Fees to Deposit Account

☒ Charge any additional fees associated with this paper or during the pendency of this application, or credit any overpayment, to deposit account

8. Deposit account number: 20-1430

DO NOT USE THIS SPACE

9. Statement and signature.

To the best of my knowledge and belief, the foregoing information is true and correct and any attached copy is a true copy of the original document.

Robert J. Bennett — Name of Person Signing
[Signature] — Signature
May 2, 1997 — Date

Atty. Reg. No. 27,533

Total number of pages including cover sheet, attachments, and document: 4

10. *Change Correspondence Address to that of Part 5?* ☒ Yes ☐ No

OMB No. 0651-0011 (exp. 4/94)

Do not detach this portion

Mail documents to be recorded with required cover sheet information to:
Asst. Commissioner for Patents
Box Assignments
Washington, D.C. 20231

06/13/1997 JSHABAZZ 00000039 DAH: 201430 08784453
Sale Ref: 00000039 DAH: 201430 08784453
01 FC:581 40.00 CH

Attorney Docket No. 010577-0404

ASSIGNMENT OF PATENT APPLICATION

JOINT

WHEREAS, **Thomas Collins** of **14890 Baranza Lane, Saratoga, California 95070**, **Dale Hopkins** of **2425 Ric Drive, Gilroy, California 95020**, **Susan Langford** of **1275 Poplar Avenue, #101, Sunnyvale, California 94086** and **Michael Sabin** of **883 Mango Avenue, Sunnyvale, California 94087**, hereinafter referred to as "Assignors," are the inventors of the invention described and set forth in the below identified application for United States Letters Patent:

Title of the Invention: **PUBLIC KEY CRYPTOGRAPHIC APPARATUS AND METHOD**

Date(s) of execution of **Declaration**: ______________________

Filing date: **January 16, 1997** Application No.: **08/784,453** ; and

WHEREAS, **TANDEM COMPUTERS INCORPORATED** a Delaware Corporation, located at 10435 North Tantau Avenue, Loc. 200-16, Cupertino, California 95014, hereinafter referred to as "Assignee," is desirous of acquiring an interest in the invention and application and in any Letters Patent and Registrations which may be granted on the same;

For good and valuable consideration, receipt of which is hereby acknowledged by Assignors, Assignors have assigned, and by these presents do assign to Assignee all right, title and interest in and to the invention and application and to all foreign counterparts (including patent, utility model and industrial designs), and in and to any Letters Patent and Registrations which may hereafter be granted on the same in the United States and all countries throughout the world, and to claim the priority from the application as provided by the Paris Convention. The right, title and interest is to be held and enjoyed by Assignee and Assignee's successors and assigns as fully and exclusively as it would have been held and enjoyed by Assignors had this assignment not been made, for the full term of any Letters Patent and Registrations which may be granted thereon, or of any division, renewal, continuation in whole or in part, substitution, conversion, reissue, prolongation or extension thereof.

Assignors further agree that they will, without charge to Assignee, but at Assignee's expense, (a) cooperate with Assignee in the prosecution of U.S. Patent applications and foreign counterparts on the invention and any improvements, (b) execute, verify, acknowledge and deliver all such further papers, including patent applications and instruments of transfer and (c) perform such other acts as Assignee lawfully may request to obtain or maintain Letters Patent and Registrations for the invention and improvements in any and all countries, and to vest title thereto in Assignee, or Assignee's successors and assigns.

IN TESTIMONY WHEREOF, Assignors have signed their names on the dates indicated.

Date: 4-30-97 Thomas W. Collins

THOMAS COLLINS

STATE OF California

COUNTY OF Santa Clara

On 30 April 1997, before me, Susan Munson, Notary Public (here insert name and title of the officer), personally appeared **Thomas Collins**, personally known to me (or proved to me on the basis of satisfactory evidence) to be the person whose name is subscribed to the within instrument and acknowledged to me that he executed the same in his authorized capacity, and that by his signature on the instrument the person, or the entity upon behalf of which the person acted, execute the instrument.

SUSAN E MUNSON
Commission #1082589
Notary Public — California
Santa Clara County
My Comm. Expires Jan 8,2000

WITNESS my hand and official seal.

Signature Susan E Munson (Seal)

Date: 4/29/97 Dale Hopkins

DALE HOPKINS

STATE OF California

COUNTY OF Santa Clara

On April 29 1997, before me, Kimberly J. Bell, Notary Public (here insert name and title of the officer), personally appeared **Dale Hopkins**, ~~personally known to me~~ (or proved to me on the basis of satisfactory evidence) to be the person whose name is subscribed to the within instrument and acknowledged to me that he executed the same in his authorized capacity, and that by his signature on the instrument the person, or the entity upon behalf of which the person acted, execute the instrument.

WITNESS my hand and official seal.

KIMBERLY J. BELL
Commission #1067882
Notary Public — California
San Mateo County
My Comm. Expires Sep 8,1999

Signature Kimberly J Bell (Seal)

Date: 4/29/97

STATE OF California

COUNTY OF Santa Clara

Susan K. Langford
SUSAN LANGFORD

On April 29, 1997, before me, Kimberly J. Bell, Notary Public (here insert name and title of the officer), personally appeared **Susan Langford**, ~~personally known to me (or~~ proved to me on the basis of satisfactory evidence~~)~~ to be the person whose name is subscribed to the within instrument and acknowledged to me that she executed the same in her authorized capacity, and that by her signature on the instrument the person, or the entity upon behalf of which the person acted, execute the instrument.

WITNESS my hand and official seal.

KIMBERLY J. BELL
Commission #1067882
Notary Public — California
San Mateo County
My Comm. Expires Sep 8, 1999

Signature Kimberly J. Bell (Seal)

Date: 30 APR 97

Michael J. Sabin
MICHAEL SABIN

STATE OF California

COUNTY OF Santa Clara

On April 30 1997, before me, Douglas M. Farnham (here insert name and title of the officer), personally appeared **Michael Sabin**, personally known to me (or proved to me on the basis of satisfactory evidence) to be the person whose name is subscribed to the within instrument and acknowledged to me that he executed the same in his authorized capacity, and that by his signature on the instrument the person, or the entity upon behalf of which the person acted, execute the instrument.

WITNESS my hand and official seal.

DOUGLAS M. FARNHAM
COMM. 1027805
NOTARY PUBLIC - CALIFORNIA
SANTA CLARA COUNTY
My Comm. Expires MAY 25, 1998

Signature Douglas M Farnham (Seal)

t\404\assign
ASSIGN.MRG 9/96